Conception des SI

Marie Lefevre

MIF17 – Automne 2012

Support de cours fortement inspiré de celui de Yannick Prié

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Qu'est-ce qu'un SI ?

- Définition (Reix, 2004)
  - Un SI
    - Ensemble organisé de ressources : matériel, logiciel, personnel, données, procédures… permettant d'acquérir, de traiter, de stocker des informations (sous formes de données, textes, images, sons, etc.) dans et entre des organisations.
  - Une organisation
    - Un ensemble d'individus : participants, acteurs ;
    - Un ou plusieurs objectifs partagés par les divers participants ;
    - Une division du travail, définissant le rôle de chaque participant ;

Exemples de SI ?

- Une application de gestion de stocks d'un supermarché
- Un site web de vente en ligne
- Une bibliothèque numérique
- Un portail avec intranet pour l'Université Lyon 1
- …

Les 3 dimensions d'un SI

- Informationnelle
  - le SI produit des représentations, manipule et produit de l'information
- Technologique
  - le SI est un construit à base d'outils, utilise les technologies de l'information
  - Rq. : système d'information ≠ système informatique
- Organisationnelle
  - le SI est un élément des processus et de la structure de l'organisation
    - SI comme source d'information externe
    - SI comme outil interactif mobilisable dans l'activité
    - SI intégré dans le système de travail

Bilan

- Système d'information
  - élément constitutif de la structure de l'organisation
- Mise en place d'un SI = choix organisationnel et technologique
- Résultat technologique issu d'un double processus de construction
  - Délibérée : conception et implantation dans l'organisation
    - génératrice de règles, de contraintes
    - offre des ressources supplémentaires (automatisation, nouvelles présentation d'information)
    - changements prévus
  - Emergente : appropriation de la technologie, assimilation, détournement
    - le résultat est indéterminé (impossible à prévoir)
    - changements imprévus

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Concevoir un SI

- Plusieurs possibilités
  - Introduire un SI dans une organisation qui n'en a pas
  - Etendre/compléter le SI d'une organisation
  - Concevoir SI et organisation en même temps
  - Informatiser le SI d'une organisation
  - …

Objectifs du concepteur

- Concevoir une application
  - Qui réponde aux besoins
    - du commanditaire
    - des utilisateurs
  - Dont on puisse prévoir à l'avance les fonctionnalités principales
  - Dont on puisse vérifier qu'elle fait bien ce qui avait été prévu
  - Capable d'évoluer, sécurisée, documentée, …

UML : *No silver bullet*

- Connaître UML, les langages de programmation ou maîtriser un AGL
  - Pas suffisant pour réaliser de bonnes conceptions
  - Il faut en plus savoir penser

- Méthodes de conception
  - propositions de cheminements à suivre pour concevoir
  - Pas de méthode ultime
  - Certains bon principes se retrouvent cependant partout…

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

Dans ce cours…

- Beaucoup de concepts
  - 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
    - pratiquer

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Génie Logiciel (Software Engineering)

- Définition (O. Boissier)
  - 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

Qualité d'un logiciel

- Point de vue utilisateur
  - Validité / Correction
    - répond aux besoins exprimés dans le cahier des charges
  - Fiabilité / Robustesse
    - capacité à fonctionner dans des conditions non prévues dans le cahier des charges ou anormales
  - Extensibilité
    - facilité avec laquelle il est possible d'ajouter des fonctionnalités au logiciel
  - Compatibilité
    - capacité à être combiné avec d'autres logiciels

Qualité d'un logiciel

- Point de vue utilisateur
  - Efficacité
    - utilisation optimale des ressources matérielles et du temps
  - Intégrité / sécurité
    - protection du code et des données, aptitude du logiciel à se protéger contre des accès non autorisés
  - Convivialité
    - Facilité d'apprentissage et d'utilisation, de préparation des données, de correction des erreurs d'utilisation, d'interprétation des retours

  => Ergonomie

Qualité d'un logiciel

- Point de vue concepteur
  - Réutilisabilité
    - aptitude à être réutilisé, même partiellement, dans d'autres applications
  - Portabilité
    - aptitude du logiciel à être transféré dans des environnements logiciels et matériels différents
  - Vérifiabilité
    - aptitude du logiciel à être testé (optimisation de la préparation et de la vérification des jeux de tests)
  - Lisibilité
  - Modularité

Notion de projet

- Définition d'un projet (Boissier)
  - 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
  - Détermine les objectifs, le budget et les délais de réalisations
- Maître d'œuvre
  - Personne physique ou morale qui reçoit mission du maitre d'ouvrage pour assurer la conception et la réalisation de l'ouvrage

Notion de projet

- Conduite de projet
  - Organisation méthodologique mise en œuvre pour faire en sorte que l'ouvrage réalisé par le maitre d'œuvre réponde aux attentes du maitre d'ouvrage dans les contraintes de temps, de cout et de 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

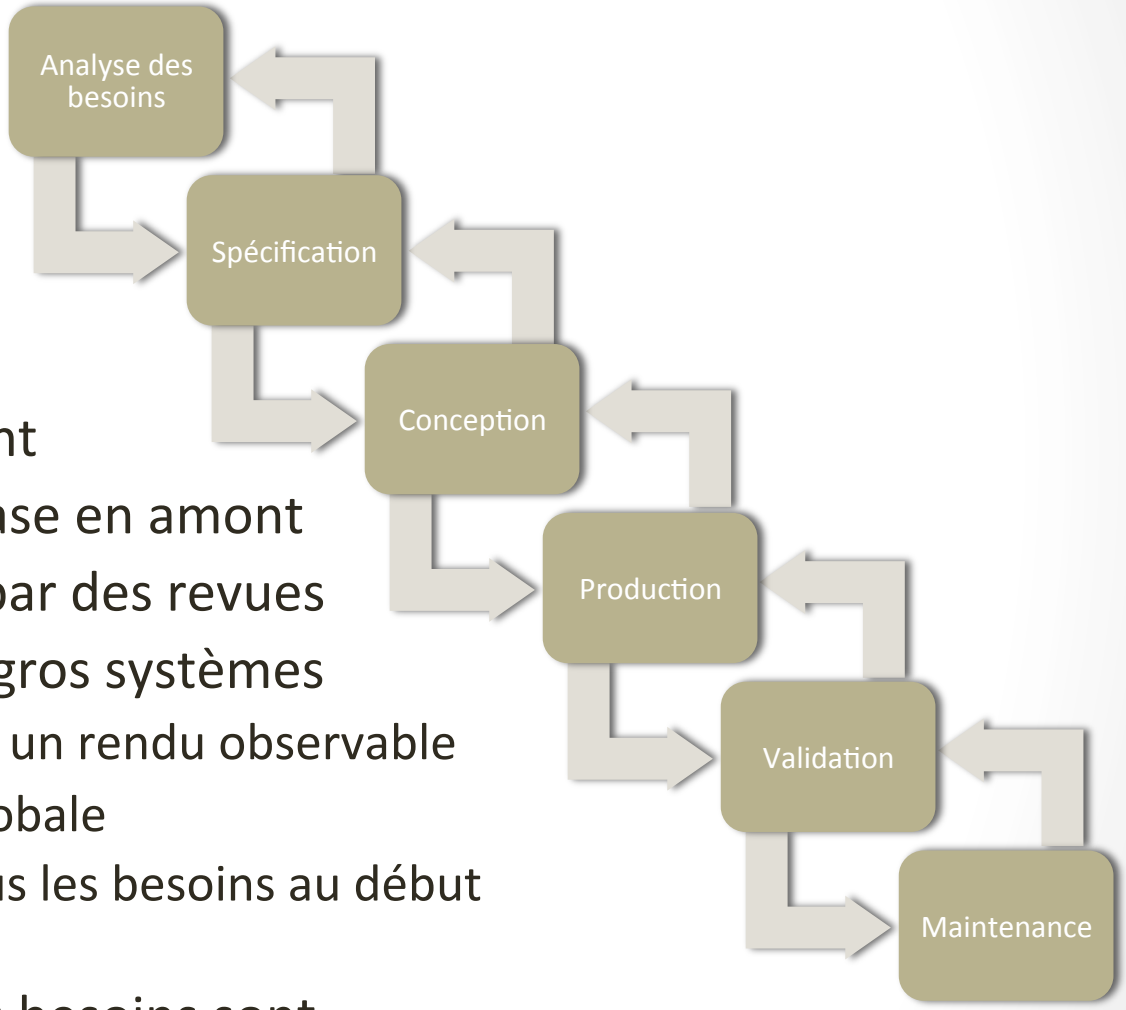
- Problème
  - Comment développer un projet de développement logiciel ?

- Solution
  - Définir et utiliser des méthodes
    - Spécifiant des processus de developpement
    - Organisant les activités du projet
    - Définissant les artefacts du projet
    - Se basant sur des modèles

Cycle de vie d'un logiciel

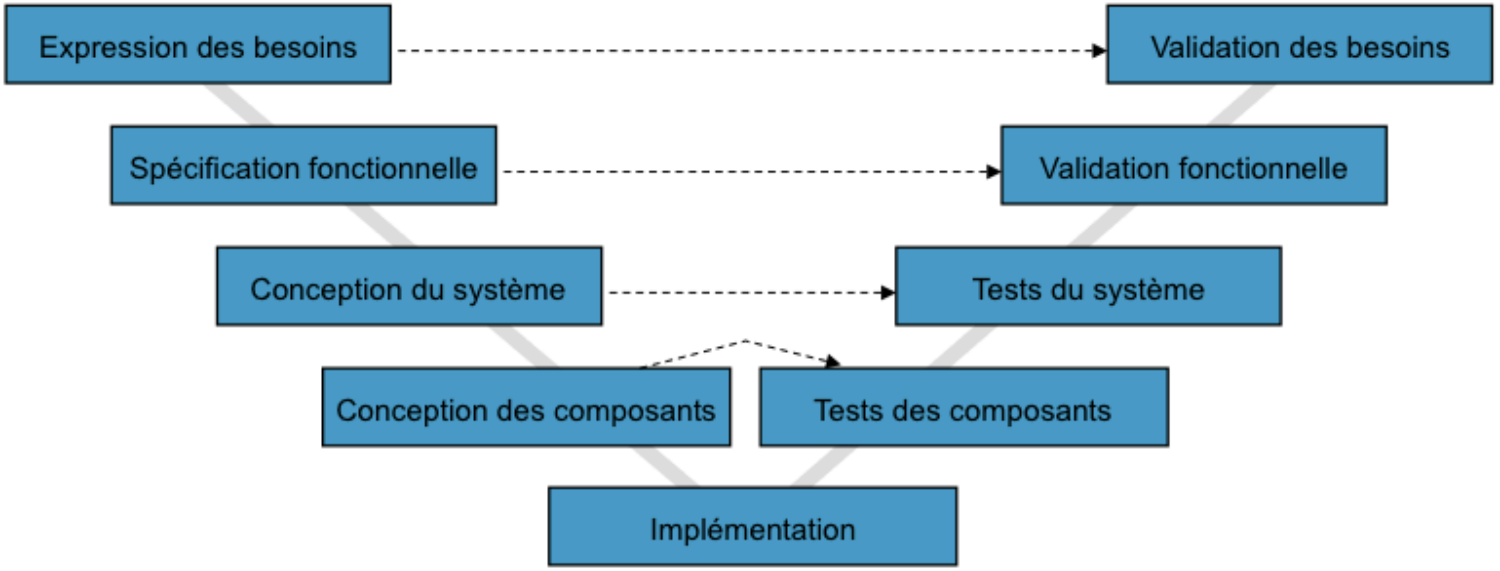
- Cycle de vie
  - Ensemble des étapes par lesquelles passe un logiciel depuis sa phase de spécification jusqu'à sa maintenance une fois en phase d'exploitation.
- Modèles de cycle de vie
  - organiser les différentes phases du cycle de vie pour l'obtention d'un logiciel fiable, adaptable et efficace
  - 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
  - Modèles 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 avoir un rendu observable
  - test de l'application globale
  - 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)

Bilan du cycle linéaire

- 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
- Non-prise en compte de l'évolution des besoins pendant le cycle
- Les études montrent (Larman 2005) :
  - 25% des exigences d'un projet sont modifiés (35-50% pour les gros projet)
  - 45% de fonctionnalités spécifiées jamais utilisées

  => 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 strictes
  - Modèle théorique parfait mais non adapté aux humains

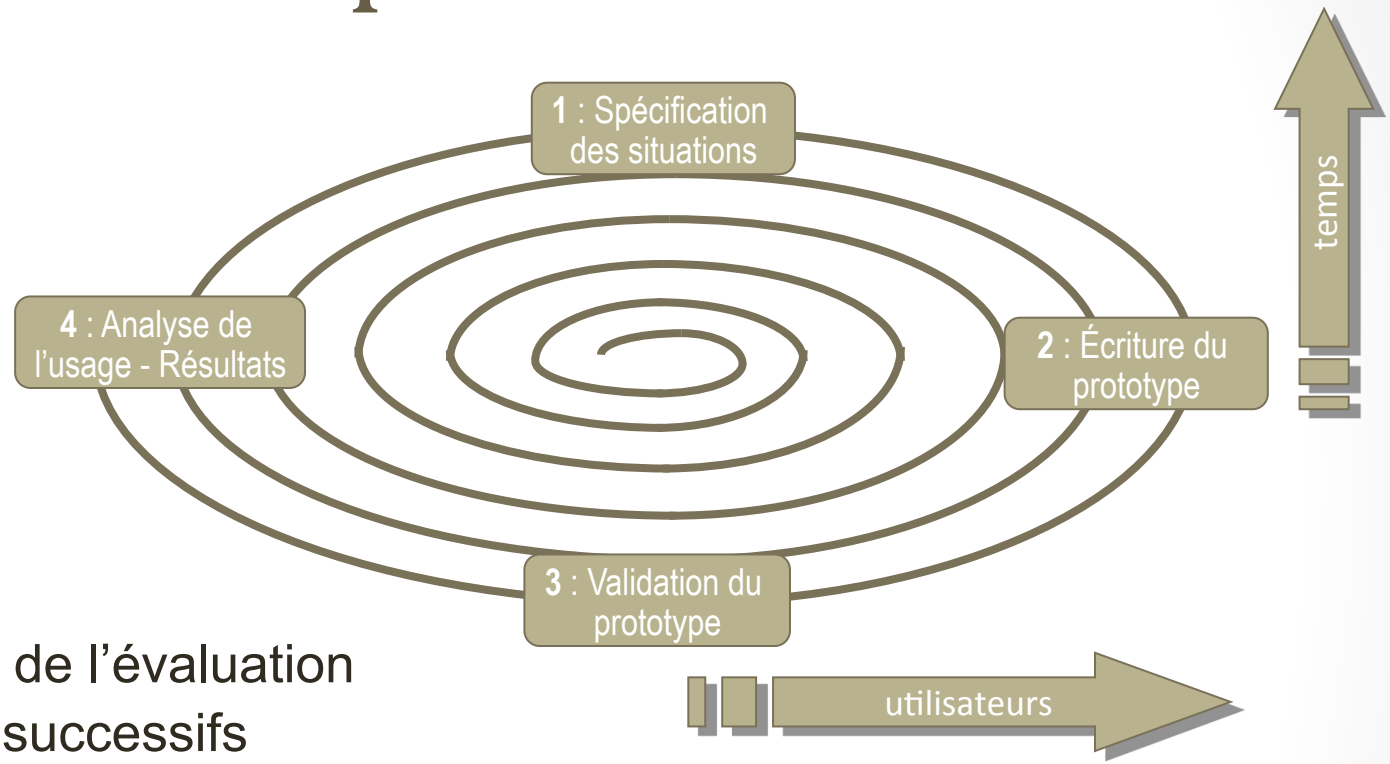
Changement de philosophie

- Nécessité de reconnaître que le changement est une constante (normale) dans les projets logiciels
  - Feedback et adaptation : décision tout au long du processus
  - Convergence vers un système satisfaisant

- Idée

  => Construction du système par incréments

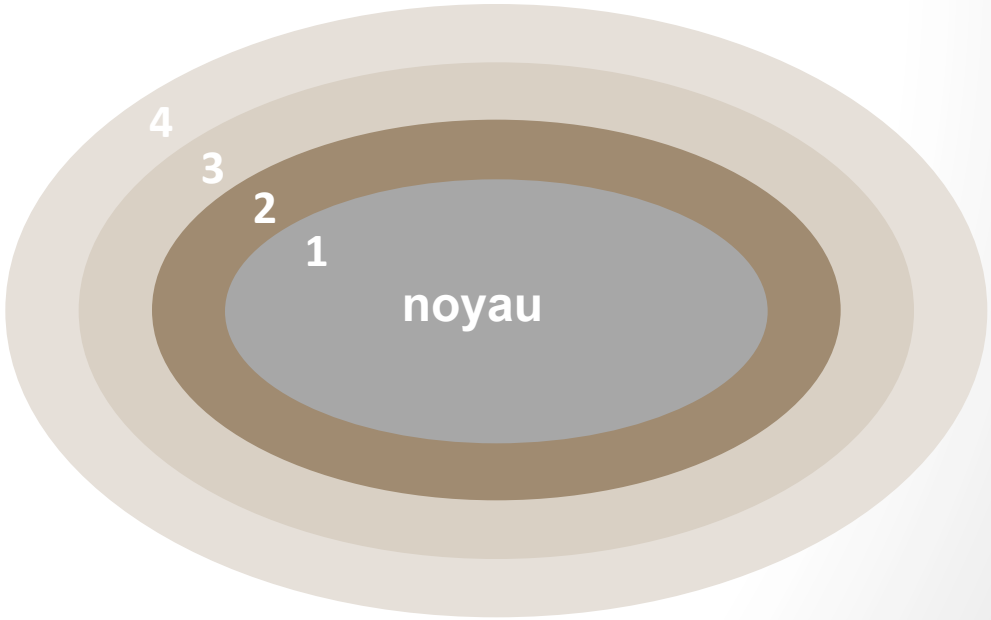
Modèle en spirale



- Importance de l'évaluation
- Prototypes successifs
- Pour chaque cycle le modèle explicite
  - l'identification des objectifs, l'alternative retenue pour atteindre les objectifs et les contraintes
  - l'analyse et la résolution des risques
  - le développement, la validation et la vérification de la phase
  - la planification de la phase suivante

Modèle par incréments

- On développe tout d'abord le noyau
- On ajoute petit à petit des fonctionnalités
- Risques
  - rencontrer un problème pour l'ajout d'un élément
  - remettre en question les éléments précédents
  - voire même le noyau



Bilan du cycle non linéaire

- Incréments successifs, itérations
- Approche souvent à base de prototypes
- Spécification des incréments difficiles
- De plus en plus difficile de modifier
- Gestion de projet pas évidente

=> Les méthodes objet en dérivent

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

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
  - 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
  - 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

- Plusieurs classes de méthodes (Bézivin)
  - organisation stratégique
  - méthodes de développement
  - méthodes de conduite de projet
  - méthode d'assurance et de contrôle qualité

- Méthode de développement
  - construire des systèmes opérationnels
  - organiser le travail dans le projet
  - gérer le cycle de vie complet
  - gérer les risques
  - gérer les coûts
  - obtenir de manière répétitive des produits de qualité constante

Processus

- Pour décrire qui fait quoi à quel moment et de quelle façon afin d'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

Artefacts d'un projet

- Tout élément d'information utilisé ou généré pendant un 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 tests
  - Programme de test
  - Interview d'un utilisateur potentiel du système
  - Description du contexte d'utilisation matériel
  - Diagramme d'architecture globale
  - Prototype
  - Rapport de réalisation
  - Modèle de dialogue
  - Rapport de qualimétrie
  - Manuel utilisateur
  - ….

Notation

- Formalisme graphique de représentation (par exemple UML)
  - pour représenter de façon uniforme l'ensemble des artefacts logiciels produits ou utilisés pendant le cycle de développement
  - pour faciliter la communication, l'organisation et la vérification

- Méthode / processus
  - Types d'artefacts + notation + démarche (+ outils)
  - Façon de modéliser et façon de travailler

Evolution 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

Origine… modélisation par les fonctions

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

- Points faibles
  - focus sur fonctions en oubliant les données
  - règles de décomposition non explicitées
  - réutilisation hasardeuse

Ensuite… 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)

Maintenant… modélisation orientée-objet

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

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Développement logiciel et activités

- Cinq grandes activités
- Emergées de la pratique et des projets

  - spécification des besoins
  - analyse
  - conception
  - Implémentation
  - tests

Activité : spécification des besoins

- Fondamentale mais difficile
- Règle d'or
  - les informaticiens sont au service du client, et pas l'inverse
- Deux types d'exigences
  - Exigences fonctionnelles
    - à quoi sert le système
    - ce qu'il doit faire
  - Exigences non fonctionnelles
    - performance, sûreté, portabilité, etc.
    - critères souvent mesurables
- Prise en compte des utilisateurs dans la « phrase de conception » à divers degrés

Besoins : modèle FURPS+

- Fonctionnality / Fonctionnalités
  - fonctions, capacité et sécurité
- Utilisability / Utilisabilité
  - facteurs humains, aide et documentation
- Reliability / Fiabilité
  - fréquence des pannes, possibilité de récupération et prévisibilité
- Performance
  - temps de réponse, débit, exactitude, disponibilité et utilisation des ressources
- Supportability / Possibilité de prise en charge
  - 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

Prise en compte des utilisateurs : Conception centrée utilisateur

- L'utilisateur et la tâche qu'il doit accomplir sont au centre de la démarche de conception
- L'utilisateur est pris en compte dès la phase d'analyse
- Relations concepteur – utilisateur
  - utilisateur
    - observé dans la résolution de sa tâche
    - interrogé sur ses attentes
    - questionné sur le logiciel conçu
- Nécessite
  - modèle de l'utilisateur
  - modèle de la tâche
  - modèle de l'interaction

Prise en compte des utilisateurs : CCU – Modèle de l'utilisateur

- Objectifs
  - identifier les caractéristiques pertinentes de l'utilisateur
    - données générales
      - taille, âge, sexe, déficiences
      - niveau de formation, habitudes culturelles
    - données liées à l'application : niveau de compétences sur le domaine, en informatique
      - débutant, occasionnel, expérimenté, expert
  - logique de fonctionnement du système ⇒ logique d'utilisation
- Difficultés
  - choisir des utilisateurs représentatifs et disponibles
  - ne pas oublier le contexte réel d'utilisation
  - expliciter les comportements, les connaissances...
- Techniques de recueil d'information auprès des utilisateurs : observation directe, entretiens, questionnaires

Prise en compte des utilisateurs : CCU – Modèle de la tâche

- Méthode
  - construire la hiérarchie de tâches du système
  - spécifier chaque tâche
  - penser aux exceptions
  - évaluer la décomposition avec l'utilisateur
- Tâche
  - un but
  - une procédure pour atteindre le but
- Tâche élémentaire
  - tâche décomposable uniquement en actions physiques op. d'E/S
- Procédure
  - un ensemble de sous-tâches
  - liées par des relation de composition
  - liées par des relations temporelles

Prise en compte des utilisateurs : CCU – Modèle de l'interaction

- Établir une correspondance directe entre
  - les objets conceptuels informatiques
  - les objets d'interaction et de présentation
- Cette correspondance doit
  - apparaître comme "naturelle"
  - s'inscrire dans une cohérence d'ensemble
    - métaphore

Prise en compte des utilisateurs : Conception participative

- Prise en compte des utilisateurs
  - pas seulement comme testeurs
  - mais aussi comme partenaires de conception
    - eux seuls connaissent vraiment les tâches
    - ils peuvent être à l'origine d'innovations
- Relations concepteur – utilisateur
  - utilisateur
    - partenaire de conception à part entière
    - participe aux choix de conception

Prise en compte des utilisateurs : Conception participative

- Avantages
  - seuls les utilisateurs connaissent la réalité des tâches
  - indispensable pour les activités mal identifiées ou peu structurées
  - facilite l'acceptation du logiciel
- Inconvénients
  - augmentation des coûts de développement
  - possibilité de contradiction entre les utilisateurs participants et les autres
  - obligation d'accepter des compromis pour satisfaire des participants, même s'ils ont tort
- Techniques de conception associées
  - scénarios, Magicien d'Oz, inspections cognitives, brainstorming, prototypes

Prise en compte des utilisateurs : Conception informative

- Prise en compte des utilisateurs
  - pas seulement comme testeur
  - sans pour autant le considérer comme partenaire de conception
  - ➢ méthode imaginée pour la conception avec des enfants
- Relations concepteur – utilisateur
  - utilisateur
    - travaille dans l'équipe de conception
    - sans participer aux choix finaux

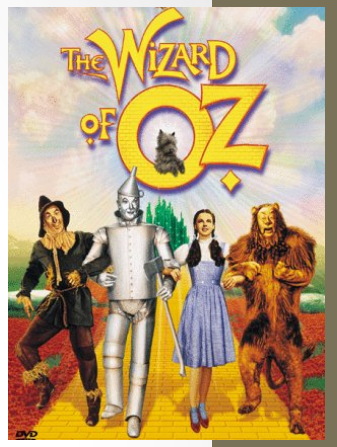
Techniques de recueil : Scénarios de conception

- But
  - créer une description réaliste de l'utilisation du nouveau système
- Moyen
  - utiliser les story-boards empruntés au monde du cinéma
  - images clés, commentaires, enchaînements
  - pour une vue d'ensemble de l'interaction
- Procédure
  - identifier des activités existantes
    - typiques
    - inhabituelles
  - créer des scénarios de travail en généralisant les histoires
    - mélanger les événements de différentes provenances
    - incorporer des situations inhabituelles dans des activités typiques
    - inclure des situations qui aboutissent et d'autres pas

Techniques de recueil : Inspections cognitives

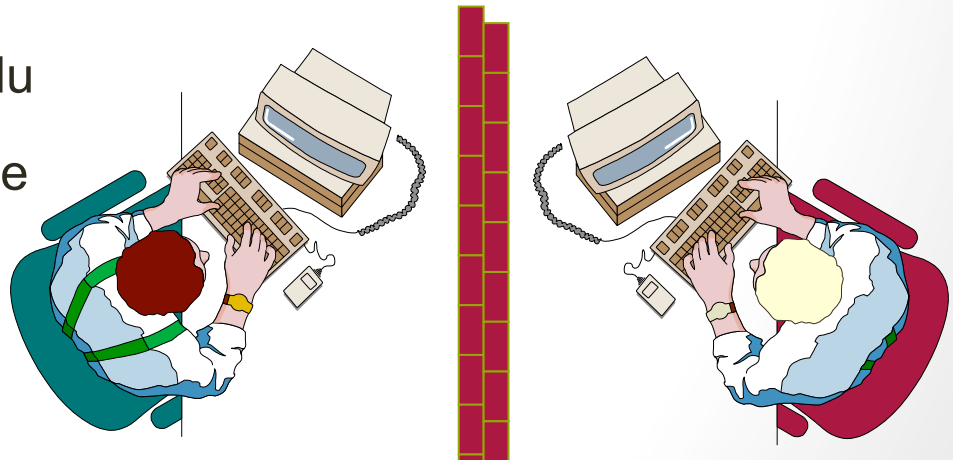
- But
  - évaluer le système en se mettant à la place de l'utilisateur
- Moyen
  - spécification d'une série de tâches et des séquences d'actions pour les réaliser
- Procédure
  - évaluation en imaginant ce que ferait l'utilisateur
    - comprend-il les messages, le comportement du système ?
  - interprétation et prise en compte des résultats

Techniques de recueil : Magicien d'Oz



Victor Fleming - 1939

- But
  - simuler les fonctionnalités absentes du système
  - système réel inexistant ou partiellement développé
  - technique difficile à mettre en place : adapté à des systèmes lourds, difficile à développer
- Moyen
  - un compère effectue les actions à la place du système
- Procédure
  - le "magicien" interprète les entrées de l'utilisateur
  - il supplée aux manques du prototype et contrôle le comportement du système
  - l'utilisateur a la sensation d'utiliser un vrai système



Techniques de recueil : Entretiens critiques

- But
  - identifier des exemples spécifiques de problèmes rencontrés par les utilisateurs
- Procédure
  - interviewer l'utilisateur dans son environnement de travail
  - lui demander de se souvenir d'un problème particulier vécu dans un passé récent
  - lui demander de décrire chaque incident en détail
  - lui demander ce qui est habituel et ce qui ne l'est pas dans l'incident

Techniques de recueil : Observations

- But
  - identifier les gros problèmes du logiciel (prototype / système final)
- Procédure
  - en laboratoire ou sur le terrain
  - choisir au moins 2 utilisateurs qui agiront indépendamment
  - définir une mission spécifique (résoudre un problème, dérouler un scénario…)
  - décider de ce que l'on veut mesurer
  - demander aux utilisateurs d'effectuer la tâche (méthode intrusive)
    - observation directe simple
    - avec explication à haute voix
    - à deux pour observer leurs interactions (interrogations, explications)
- enregistrer les interactions, puis les analyser
  - papier, audio, vidéo, trace informatique

Techniques de recueil : Questionnaires

- But
  - résumer économiquement l'avis de nombreux utilisateurs
- Procédure
  - déterminer le public (représentatif) destinataire du questionnaire
  - comment l'envoyer et le récupérer
  - comment analyser les résultats (automatiquement/ manuellement)
- Style de questions
  - informations générales
  - questions ouvertes
  - questions dirigées
  - choix multiples
  - réponses sur une échelle
  - classements

Techniques de recueil : Brainstorming

- But
  - générer un grand nombre d'idées créatives
- Moyen
  - demander aux participants d'imaginer des solutions
- Procédure
  - réunir un petit groupe avec différents rôles et expertises
  - limiter le temps (1h)
  - décrire un problème de conception spécifique
  - phase 1 : générer une grande quantités d'idées
    - faire participer tt le monde, enregistrer ttes les idées sans les évaluer
  - phase 2 : classer les idées en fonction de leur qualité
    - chacun annonce les idées qu'il préfère
    - les idées sont classées par nombre de votes
    - commencer la conception à partir des idées les mieux classées
    - ne pas oublier les idées insolites

Activités : implémentation / tests

- Implémentation
  - Dans un ou plusieurs langage(s)
  - Activité la plus coûteuse
- Tests
  - Deux grandes catégories de tests
    - Les tests fonctionnels :
      - Objectif : vérifier le respect des spécifications
      - Exemples : vérification de la correction, tests de qualité, tests de performances, etc.
    - Les tests structurels :
      - Objectif : détecter les erreurs d'implémentation
      - Exemples : débordement de pile, échec d'initialisation, résultat erroné, etc.

Quelques tests…

- Tests unitaires
  - fonction, classe, module, composant
- Tests d'intégration
  - test de composition entre plusieurs composants
- Tests de validation (ou de conformité)
  - adéquation aux spécifications
- Tests de non-régression
  - vérification que l'évolution du code ne crée pas de nouvelles anomalies
  - ce qui était valide à un moment doit le rester toujours
- Test de bon fonctionnement (test-to-pass)
  - les cas de tests contiennent des données d'entrée valides
- Test de robustesse (test-to-fail)
  - les cas de tests correspondent à des données d'entrée invalides
- Tests de performances
  - test de charge ou de stress, définissant la capacité à répondre à une demande de ressources anormale

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

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
  - Interopérables
  - 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
- Ateliers de développement logiciel
- Outils de vérification
- Outils de communication
- …

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
  - Génie Logiciel
  - Méthodes
  - Activités
  - Outils
  - Documentation de projet
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Cahier des charges

- Spécification d'un système à réaliser
  - Fonctionnalités, description du contexte, contraintes de délais, de prix, préférences…
- Document très important
  - Permet de se mettre d'accord en interne sur le système à construire
  - Permet de lancer l'appel d'offre
  - Est une partie du contrat entre le client et le prestataire
  - Sert de base et de guide au chef de projet
  - …

Cahier des charges fonctionnel

- Préciser les orientations et le champ du domaine étudié
- Analyser l'existant
  - Au niveau organisation
  - Documents utilisés
  - Traitements effectués
  - Données manipulées
- Proposer des solutions d'organisation
  - Fonctionnelles et techniques
  - Répondant aux exigences et besoins exprimés

Cahier des charges fonctionnel

- Obtenir une description globale du système
  - Organisationnelle
  - Fonctionnelle
  - Technique
  - Contraintes majeures de sécurité
  - Contraintes de performance
  - Interface avec d'autres systèmes
  - …
- Vérifier la faisabilité
  - Organisationnelle et technique
- Aboutir à un choix argumenté d'une solution type de développement

Spécification fonctionnelles

- Besoins fonctionnels métier
  - Liste des fonctionnalités que le système devra avoir
  - Scénarios
  - Cas d'utilisation
  - …
- Ne pas parler de réalisation technique

Spécification techniques

- Comment réaliser les choses
  - Liste des fonctions à coder
  - Architecture
  - …
- Attention
  - Le mélange technique / métier est facile à faire !!!

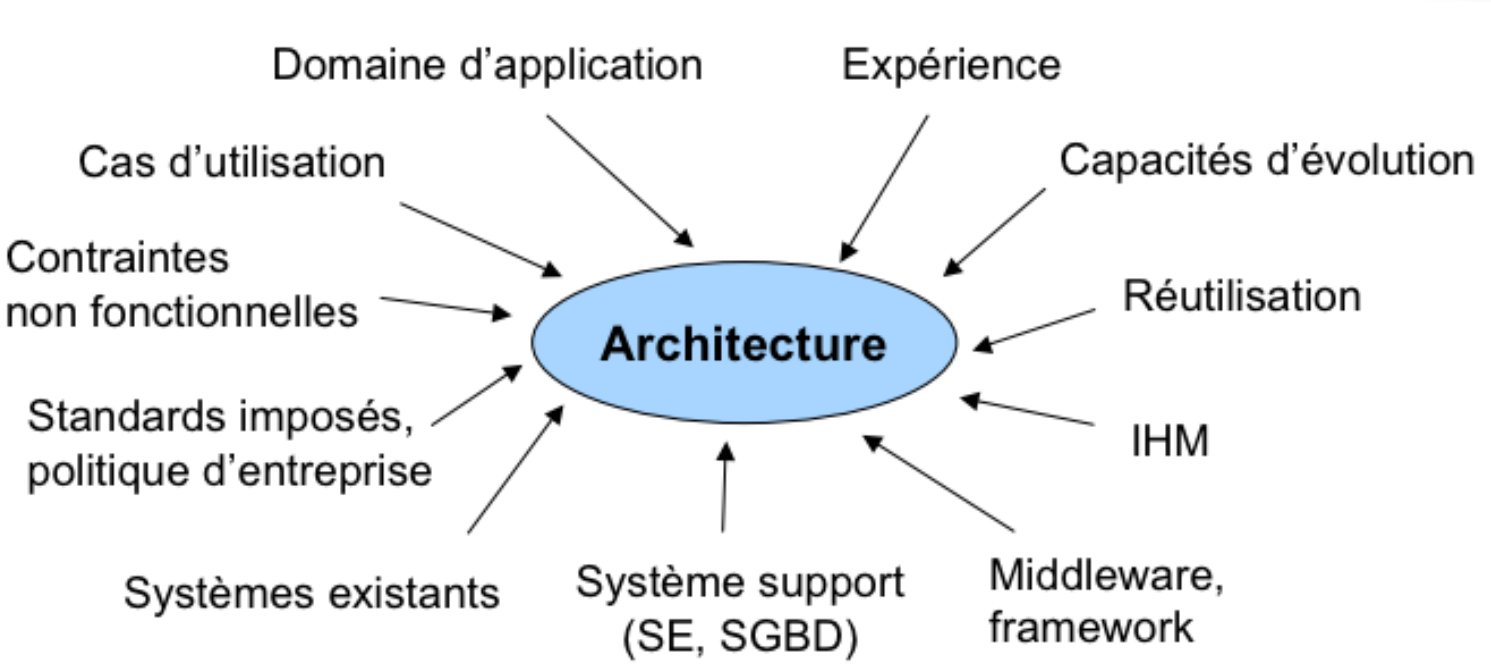
Architecture

- Difficile à définir
  - Exemple dans le bâtiment : plombier, électricité, peintre ne voient pas la même chose…
- Multiples 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, 1998)
  - *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)

Architecture

- Définition pour ce cours
  - Ensemble des décisions prises, dépendantes de contraintes, concernant
    - L'organisation du système
    - Les éléments qui structure le système
    - La composition de sous-systèmes en système
    - Le style architectural guidant l'organisation
      - Architecture en couches, centrée sur les données…
  - Ensemble des éléments de modélisation les plus signifiants qui constituent les fondations du système

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…

Axes pour considérer l'architecture

- Architecture logicielle (ou architecture logique)
  - Organisation à grande échelle des classes logicielles en packages, sous-systèmes et couches
    - 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 à base de composants
- Architecture de déploiement
  - Décision de déploiement des différents éléments
  - Déploiement des fonctions sur les postes de travail des utilisateurs
    - Par exemple, en entreprise : central/départemental/local
- Existence de patterns architecturaux
  - Couches, MVC…

Description de l'architecture

- L'architecture doit être une vision partagée
  - sur un système très complexe
  - permet de guider le développement
  - doit rester compréhensible

- Il faut donc mettre en place une description (ou documentation) explicite de l'architecture
  - description qui sert de référence jusqu'à la fin du cycle et après
  - description qui doit rester aussi stable que l'architecture de référence

Description de l'architecture

- Comment décrire l'architecture ?
  - Décrire les facteurs qui influencent l'architecture
    - Facteurs architecturaux
  - Décrire les choix qui ont été faits
    - Mémos techniques
  - Décrire l'architecture
    - Documentation d'architecture du logiciel

Description de l'architecture : Facteurs architecturaux

- Facteurs qui ont une influence significative sur l'architecture
  - Fonctionnalités
  - Performance
  - Fiabilité
  - facilité de maintenance
  - Implémentation
  - Interface
  - etc.

Description de l'architecture : Facteurs architecturaux

- Un facteur architectural doit être identifié et décrit dans une fiche (Larman 2004)

  - Nom
    - Ex. « fiabilité, possibilité de récupération »
  - Mesures et scénarios de qualité
    - Ce qu'il doit se passer et comment le vérifier
    - Ex. « si problème, récupération dans la minute »
  - Variabilité
    - Souplesse actuelle et évolutions futures
    - Ex. « pour l'instant, service simplifiés acceptable en cas de rupture. Evolution : services complets »
  - Impacts
    - Pour les parties prenantes
    - Ex. « fort impact, rupture de service non acceptable »
  - Priorité
  - Difficulté ou risque

Description de l'architecture : Mémos techniques

- Les choix architecturaux doivent prendre en compte les facteurs architecturaux
- Il est important de décrire les solutions choisies et leur motivation
  - Assurer la traçabilité des décisions architecturales
    - Raisons des choix
    - Alternatives étudiées
- Les mémos techniques décrivent les choix architecturaux
  - Texte + diagrammes

Description de l'architecture : Mémos techniques

- Structure d'un mémo technique (Larman 2004) :

  - Nom du problème étudié
  - Résumé de la solution choisie
  - Facteurs architecturaux concernés
  - Solution
  - Motivation
  - Problèmes non résolus
  - Autres solutions envisagées

Description de l'architecture

- Une vue architecturale (Larman 2004)
  - Une 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 : Document d'architecture logiciel

- DAL = récapitulatif des décisions architecturales
  - Résumé rapide de l'architecture
    - 1 diagramme + texte
  - Ensemble des facteurs architecturaux
    - Quels sont les points qui ont une influence importante sur les choix d'architecture ?
  - Ensemble des vues architecturales
    - Description du matériel et du logiciel.
  - Mémos techniques
    - Quelles sont les décisions qui ont été prises et pourquoi.

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Unified Software Development Process / Unified Process (UP)

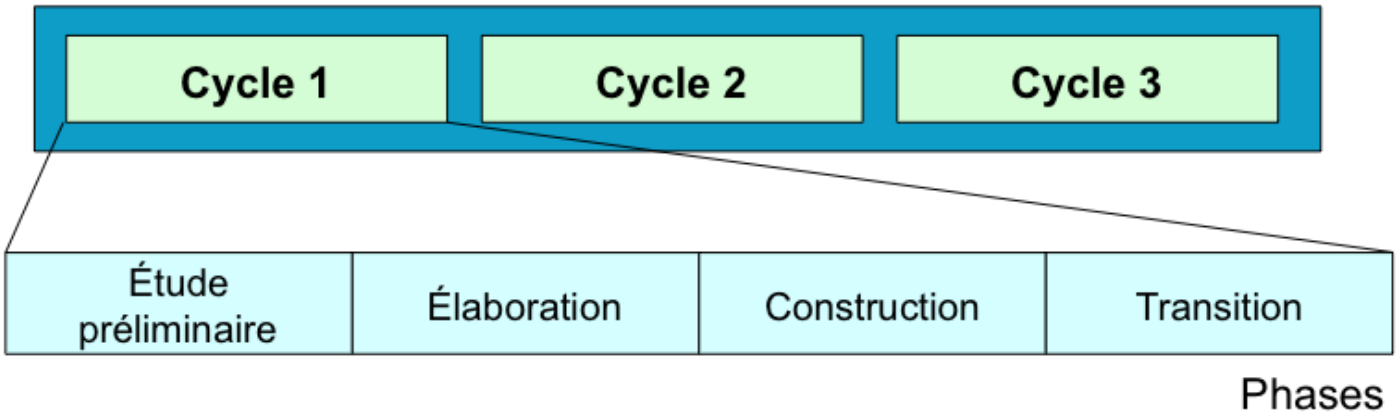
- Années 1990 : beaucoup de méthodes
  - liées à des outils, à l'adaptation à UML (comme langage de notation) de méthodes pré-existantes , aux entreprises, etc.
  - …finalement, autant de méthodes que de concepteurs / projets
- USDP
  - Proposé par Rumbaugh, Booch, Jacobson
    - les concepteurs d'UML
  - purement objet
  - prend de la hauteur par rapport à l'instanciation par RUP de Rational Software – IBM
  - 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

Un cycle de vie contient 4 phases

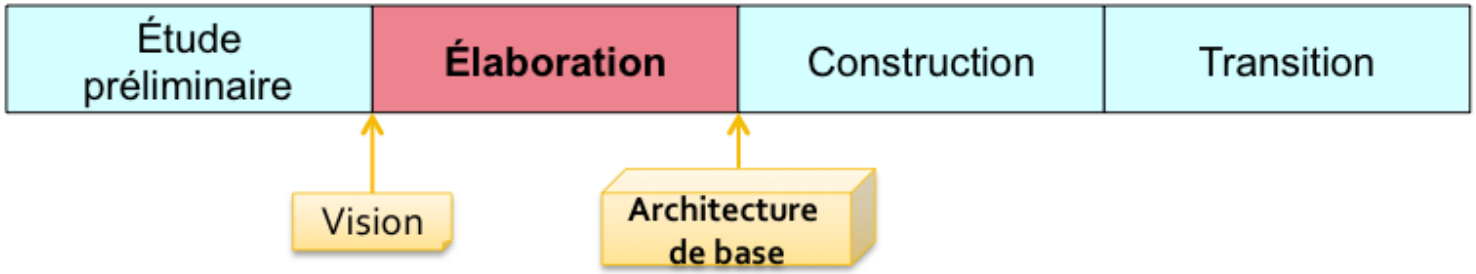


- Considérer un produit logiciel quelconque par rapport à ses versions
  - un cycle produit une version
- Gérer chaque cycle de développement comme un projet ayant quatre phases
  - vue gestionnaire (manager)
  - chaque phase se termine par un point de contrôle (ou jalon permettant aux chefs de projet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

Phase 1 : étude préliminaire

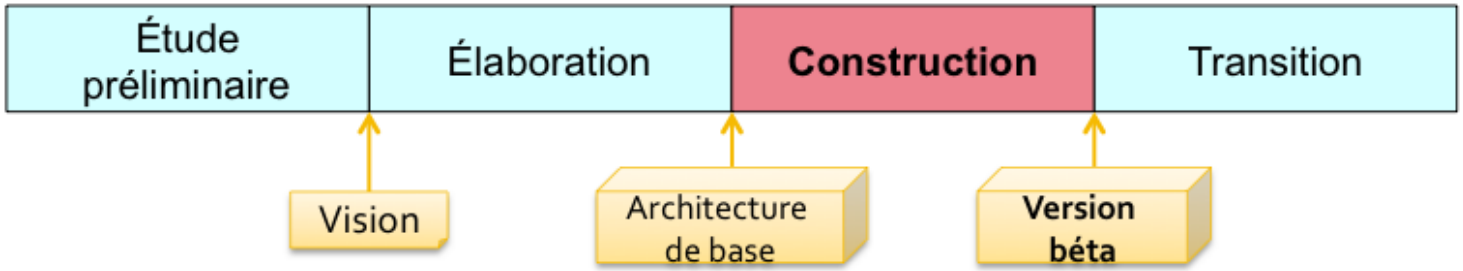
- Déterminer
  - ce que fait le système ?
  - à quoi pourrait ressembler l'architecture ?
  - quels sont les risques ?
  - quel est le coût estimé du projet ? Comment le planifier ?
- Jalon
  - « vision du projet » = document
  - Accepter le projet ?
- Coût faible

Phase 2 : élaboration



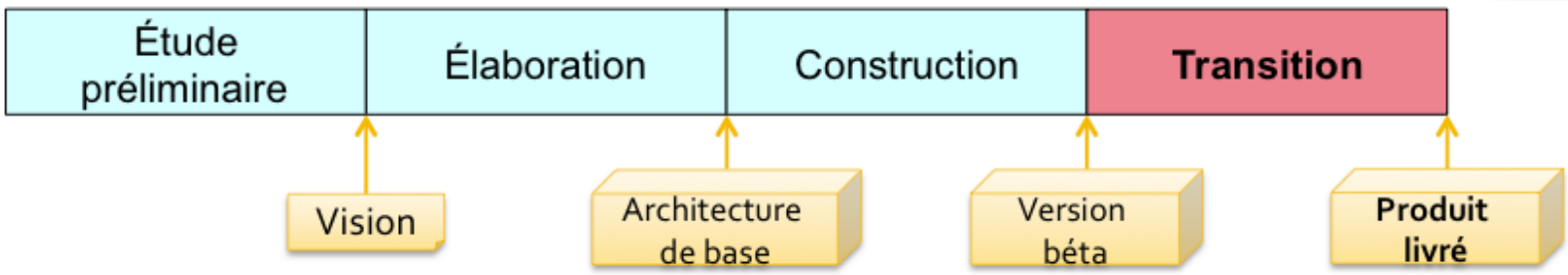
- Spécifier la plupart des cas d'utilisation
- Concevoir l'architecture de base (squelette du système)
- Mettre en œuvre cette architecture
  - CU critiques, <10 % des besoins
- Faire un planification complète
- Jalon
  - « architecture du cycle de vie » = premier prototype qui démontre la validité des choix architecturaux
  - Peut-on passer à la construction ?
    - besoins, architecture, planning stables ? Risques contrôlés ?
- Coût faible

Phase 3 : construction



- Développer par incréments
  - architecture stable malgré des changements mineurs
  - le produit contient tout ce qui avait été planifié
  - il reste quelques erreurs
- Jalon
  - « capacité opérationnelle initiale » = version béta
  - Le produit est suffisamment correct pour être installé chez un client ?
- Phase la plus couteuse
  - > 50% du cycle
  - Englobe conception, code, tests, intégration…

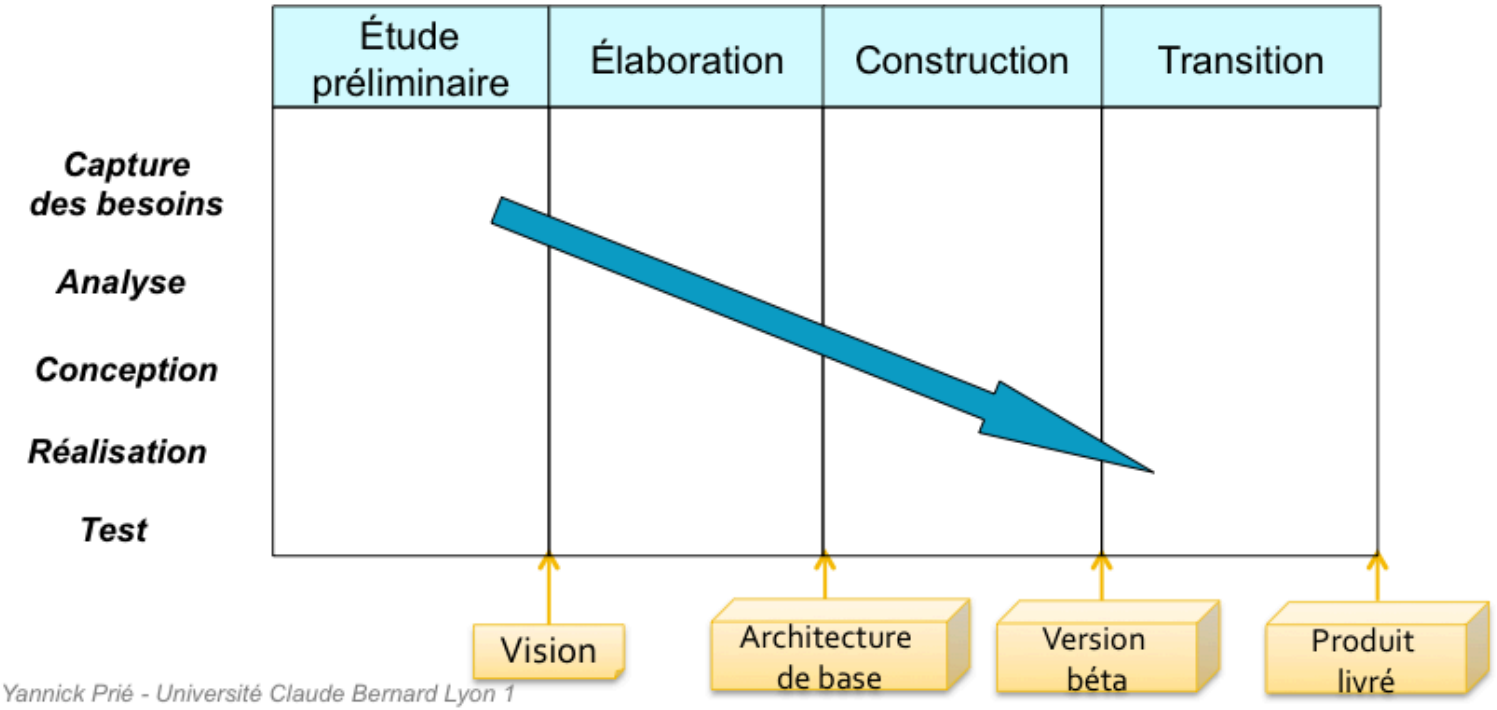
Phase 4 : transition



- Transition
  - Livrer / déployer le produit
  - Corriger le reliquat d'erreurs
  - Améliorer le produit
  - Former les utilisateurs
  - Mettre en place l'assistance en ligne
- Jalon
  - « livraison du produit » = produit déployé chez le client
    - tests suffisants ? Produit satisfaisant ? Manuels prêts ?

Phases et activités

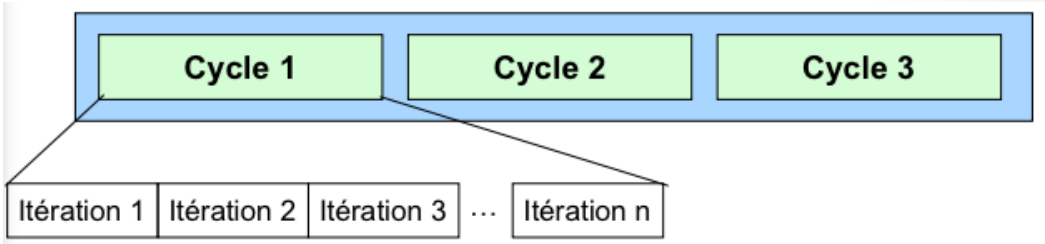
- Un cycle met en jeu des activités
  - Vue du développement sous l'angle technique (développeur)
  - Les activités sont réalisées au cours des phases



Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
    - Itératif et incrémental
    - Piloté par les besoins
    - Piloté par les risques
    - Centré sur l'architecture
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Itérations et incréments

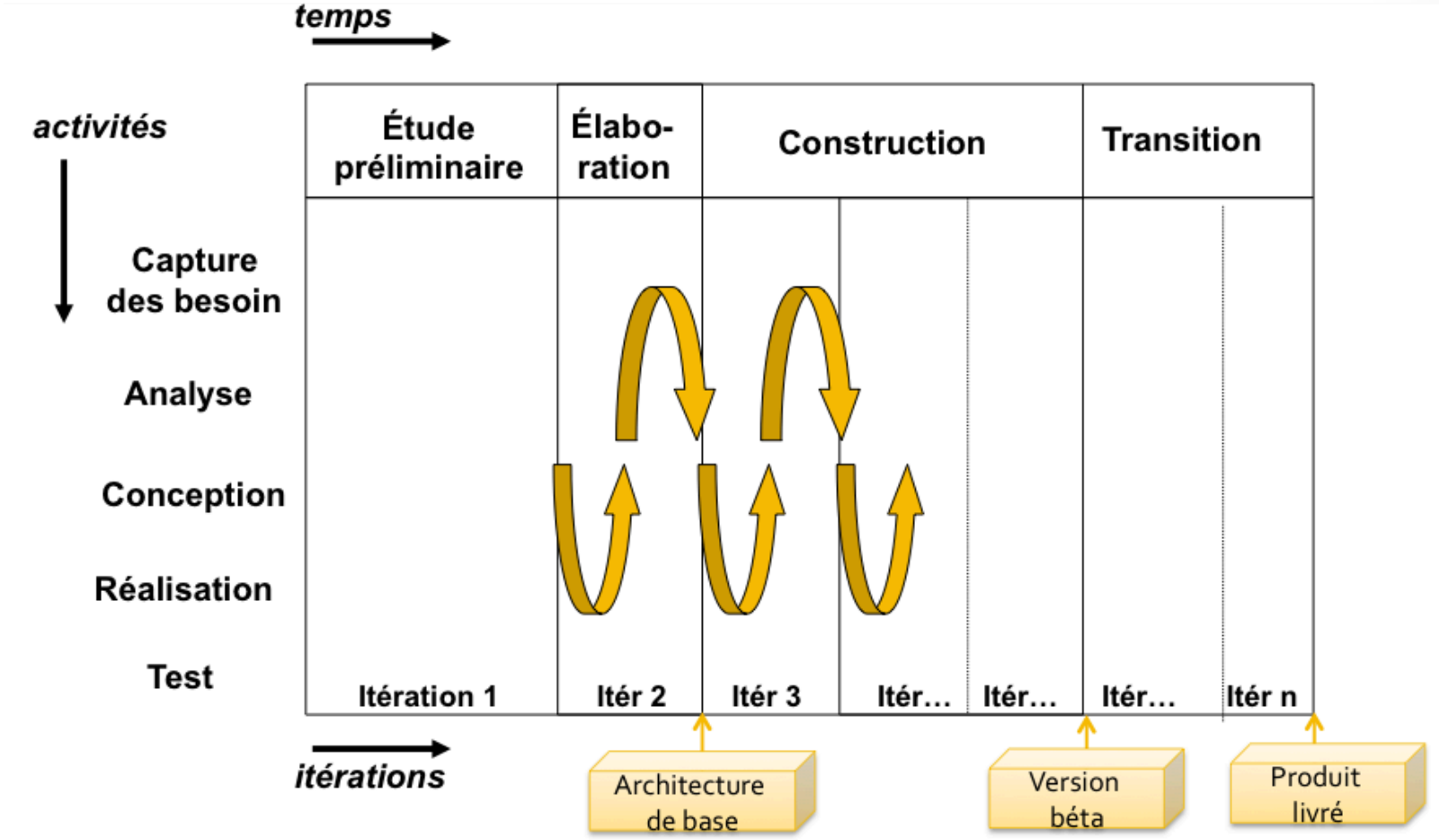


- Anti-cascade, spirale
- Construction progressive du système
  - Processus incrémental
- Chaque itération permet
  - De maîtriser une partie des risques
  - Apporte une preuve tangible de faisabilité
    - Produit un système partiel opérationnel (exécutable, testé et intégré) avec une qualité égale à celle du produit fini (alpha)
    - Système qui peut être testé : permet de savoir si on va dans la bonne direction
- Série de prototypes qui vont en s'améliorant
  - De plus en plus de fonctionnalités disponibles
    - Retours des utilisateurs

Itération

- Une itération
  - = 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 agrées, décisions pour les itérations suivantes
  - Conduit à une version montrable implémentant un certain nombre de cas d'utilisation
  - Dure entre quelques semaines et 9 mois (au delà : danger)
    - Butée temporelle qui oblige à prendre des décisions

Itérations et phases



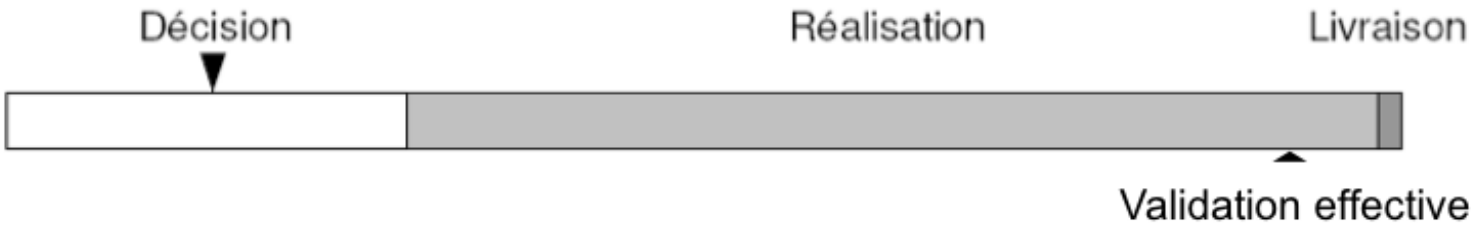
Avantage d'un processus incrémental

- Gestion de la complexité
  - pas tout en même temps, étalement des décisions importantes
- Maîtrise des risques élevés précoce
  - architecture mise à l'épreuve rapidement (prototype réel)
  - diminution de l'échec
- 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…

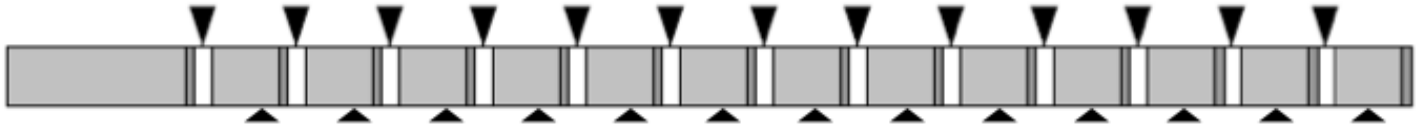
- Mais gestion de projet plus complexe : planification adaptative

Résumé

- Démarche traditionnelle



- Méthodes itératives



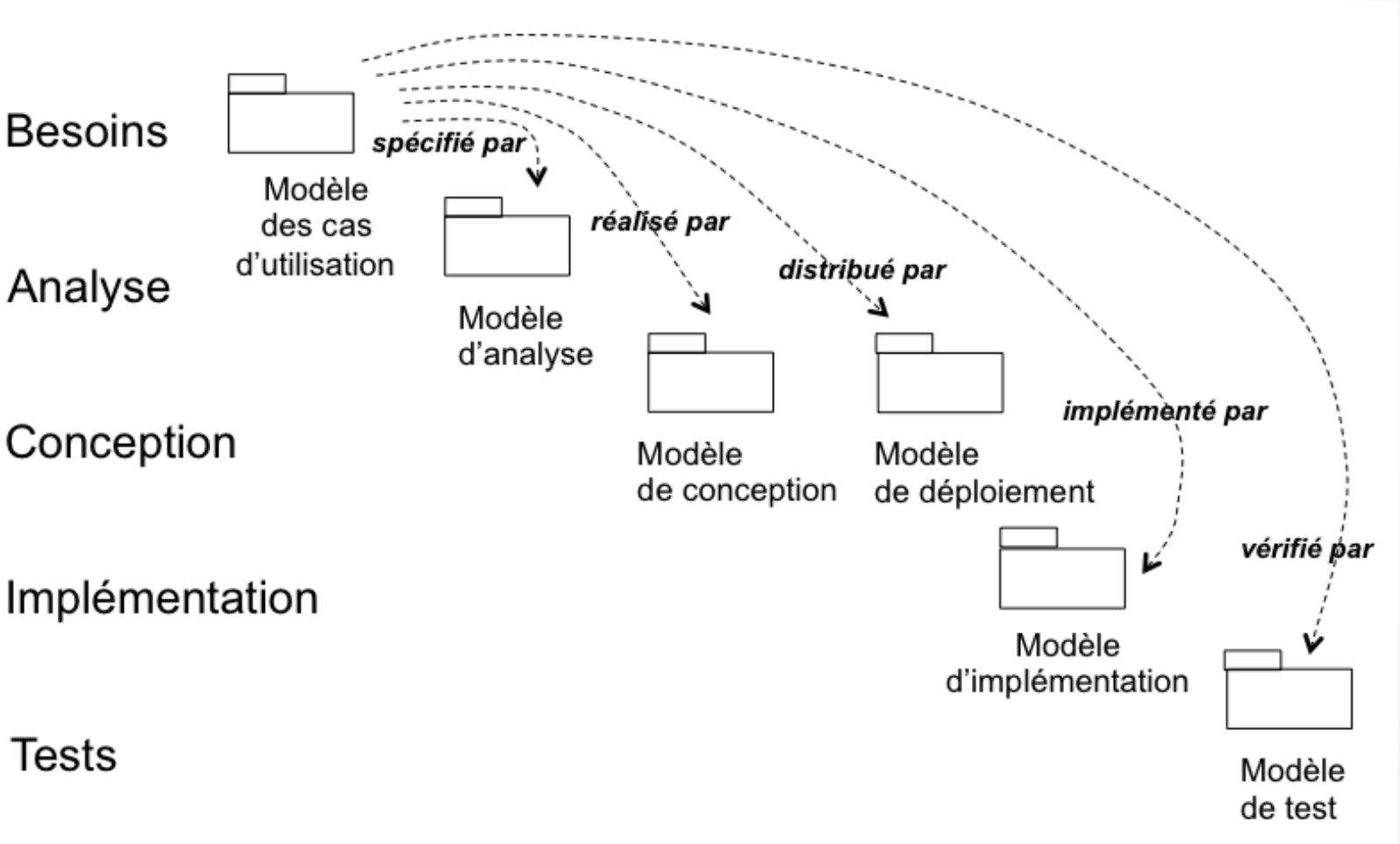
Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
    - Itératif et incrémental
    - Piloté par les besoins
    - Piloté par les risques
    - Centré sur l'architecture
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

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
    - que fait le système, pour qui, dans quel environnement ?
  - utilisation 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 CU

- Centrés utilisateurs
  - support de communication en langue naturelle entre utilisateurs et concepteurs basé sur les scénarios (et non liste de fonctions)
  - dimension satisfaction des objectifs utilisateur
    - également pour les utilisateurs informaticiens (administration)
  - besoins fonctionnels, pour acteurs humains et non humains à identifier précisément
- Assurent la traçabilité par rapport aux besoins de toute décision de conception sur l'ensemble du projet
  - tout modèle peut / doit se référer à un CU
- Fournissent une vision commune des membres du projet sur l'architecture
  - Management, conception, qualité, marketing, etc.

- Attention
  - créer de bons CU est un art
  - danger de décomposition fonctionnelle des CU qui reviendrait à une conception fonctionnelle à l'ancienne

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
    - Itératif et incrémental
    - Piloté par les besoins
    - Piloté par les risques
    - Centré sur l'architecture
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Qu'est-ce qu'un risque ?

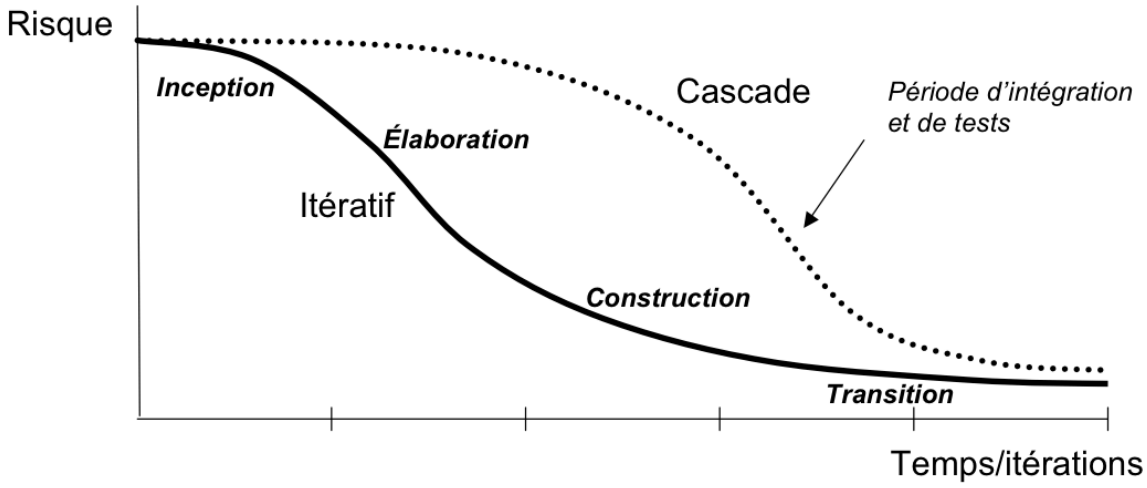
- Risque que le projet de construction du système soit un echec
- Nature des risques
  - besoins / technique / autres
- Exemples
  - le système construit n'est pas le bon
  - architecture inadaptée, utilisation de technologies mal maîtrisées, performances insuffisantes
  - personnel insuffisant, problèmes commerciaux ou financiers (risques non techniques, mais bien réels)

Gestion des risques

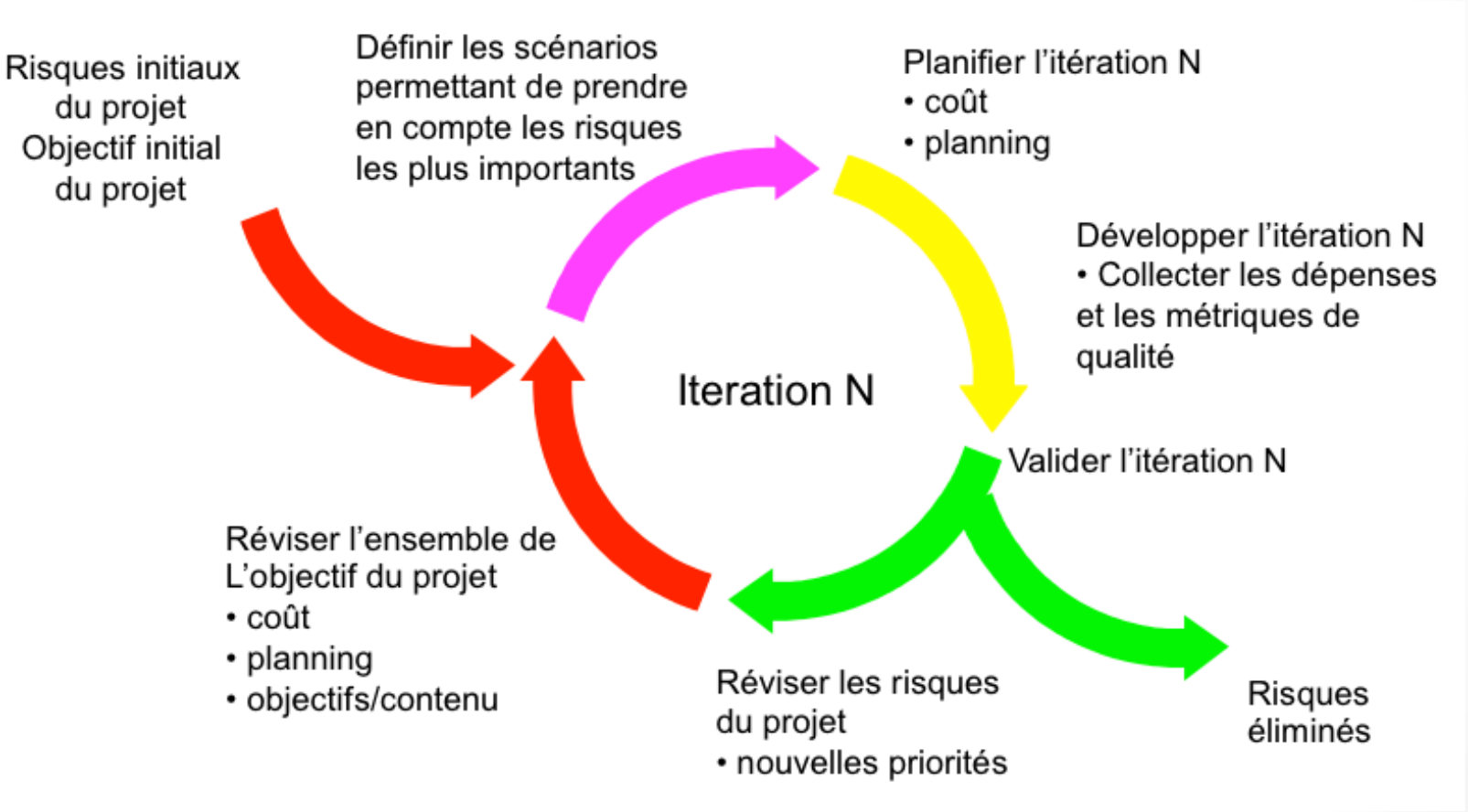
- Identifier et classer les risques par importance
- Agir pour diminuer les risques
  - Ex. changer les besoins, confiner la portée à une petite partie du projet, faire des tests pour vérifier leur présence et les éliminer
- S'ils sont inévitables, les évaluer rapidement
  - tout risque fatal pour le projet est à découvrir au plus tôt

Pilotage par les risques et itérations

- 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



Itérations pilotées par la réductions des risques



Itérations et risques

- On ordonne les itérations à partir des priorités établies par les CU et 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 au court du temps

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 : rétrospective
    - éléments à conserver, problèmes, éléments à essayer

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
    - Itératif et incrémental
    - Piloté par les besoins
    - Piloté par les risques
    - Centré sur l'architecture
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Processus centré sur l'architecture

- L'architecture sert de lien pour l'ensemble des membres du projet
  - réalisation concrète de prototypes incrémentaux qui « démontrent » les décisions prises
  - Vient compléter les CU comme « socle commun »
- Contrainte de l'architecture
  - 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

Objectif / architecture

- Construire une architecture
  - Comme forme dans laquelle le système doit s'incarner
    - les CU réalisés doivent y trouver leur place / la réalisation
    - la réalisation des CU suivant doit s'appuyer sur l'architecture
  - Qui permet de promouvoir la réutilisation
  - Qui ne change pas trop
    - converger vers une bonne architecture rapidement

Principe de construction de l'architecture

- Première étape :
  - Choisir une architecture de haut-niveau
  - Construire 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

Principe de construction de l'architecture

- Deuxième étape :
  - Construire 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 des points d'évolution les plus probables

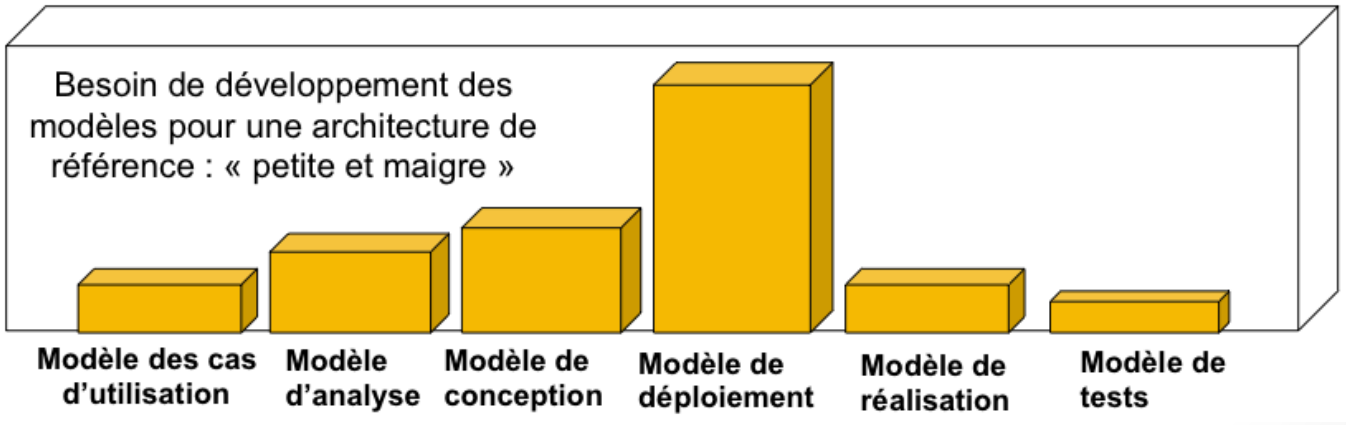
Principe de construction de l'architecture

- Dernière étape :
  - Réaliser incrémentalement les cas d'utilisation
    - Itérations successives
    - L'architecture continue à se stabiliser sans changement majeur
- Remarque
  - Maturation des cas d'utilisation
    - Plus faciles à exprimer et plus précis quand un début d'application est disponible

Architecture & phase d'élaboration

- L'architecture est conçue et réalisée pendant la phase d'élaboration
- Phase d'élaboration
  - aller directement vers une architecture robuste, à coût limité

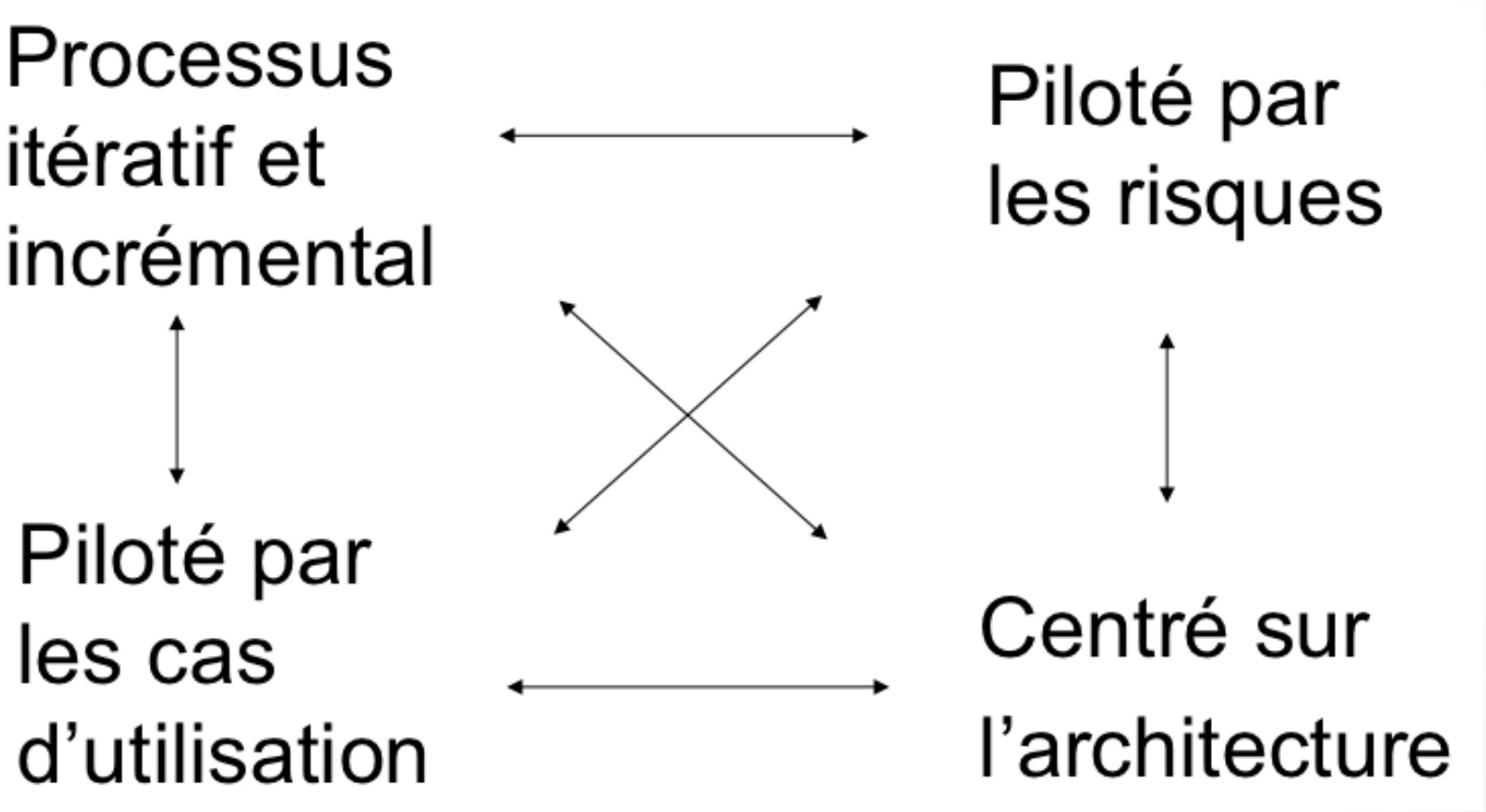
  ⇒architecture de référence
  - 10% des classes suffisent
    - ce qui est à l'intérieur des sous-systèmes n'est pas pertinent
- L'architecture de référence
  - permettra d'intégrer les CU incrémentalement pendant la construction
  - guidera le raffinement et l'expression des CU pas encore détaillés



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

Les 4 grandes caractéristiques du Processus Unifié sont liées



Plan

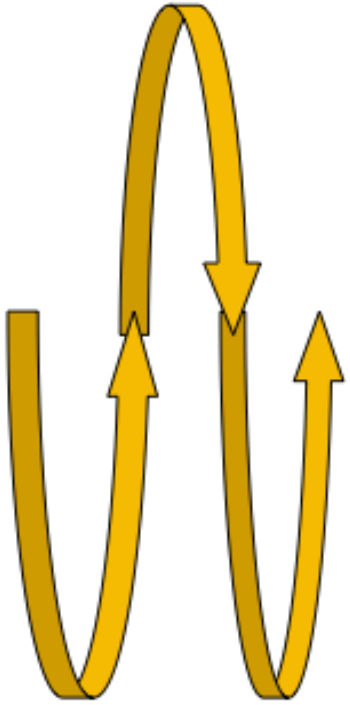
- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
    - Activités, travailleurs, artefacts & modèles
    - Différentes activités pour passer des besoins au code
    - Différentes phases pour piloter les activités
    - Quelques remarques
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Rappel : principes de conception objet

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

Principe général du PU

1. Décrire le modèle du domaine
   - Fournira l'ossature du diagramme de classe
2. Décrire les cas d'utilisation
   - Besoins fonctionnels
3. Réaliser les cas d'utilisation un par un
   - Avec des interactions entre objets
     - Diagramme de séquence / communication
4. Compléter le diagramme de classe
5. Coder les classes
   - Compléter le prototype
6. Tester et déployer

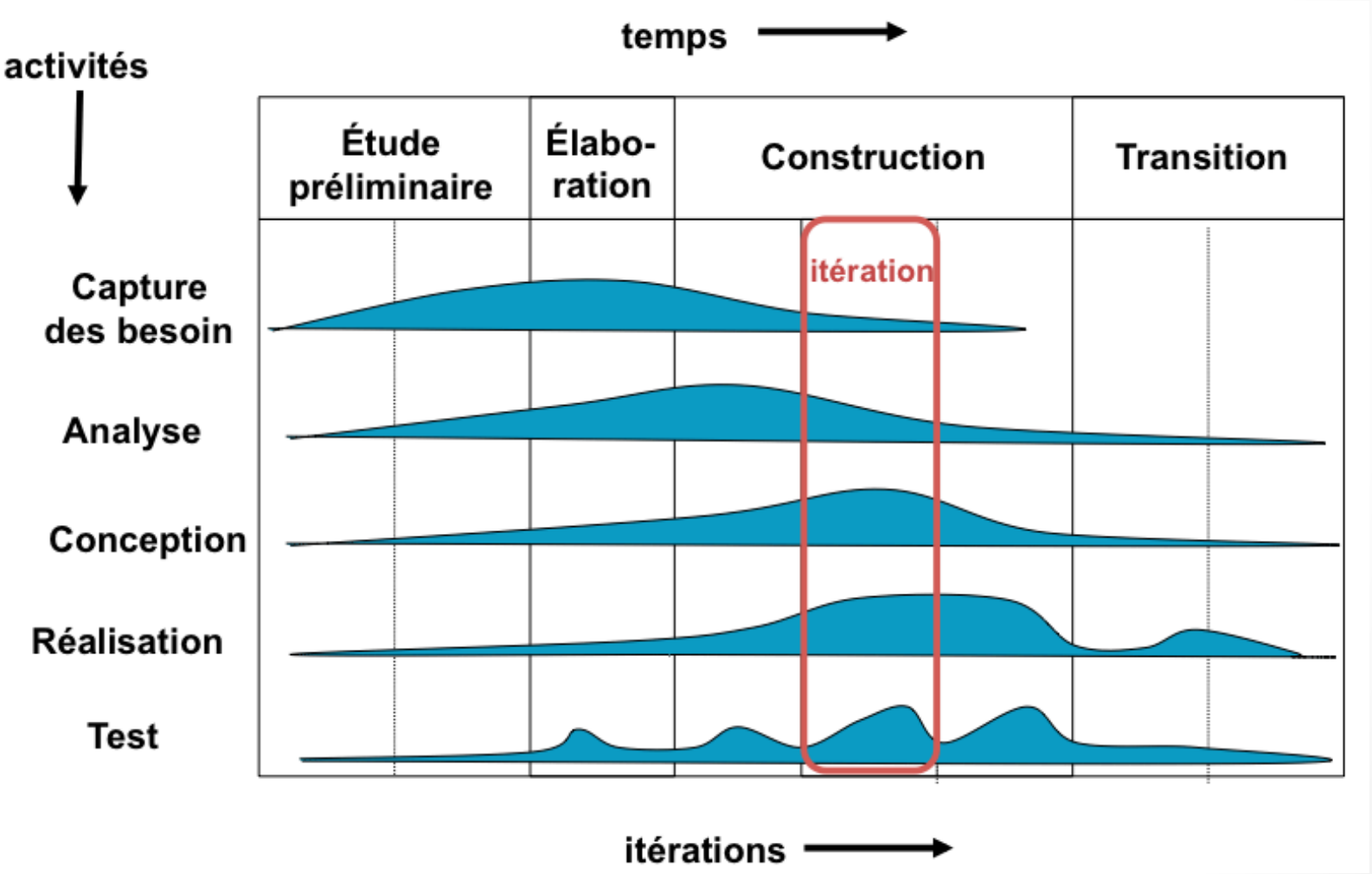


PU : activités et phases

- Activités : description technique de la méthode
  - que faut-il faire ?
  - sous quelle forme (modèles, documents textuels…) ?
  - comment obtenir les produits ?

- Phases : description du déroulement du projet
  - 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 ?

Les activités selon les phases



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 adapté au contexte du projet (taille, personnels, entreprise, compréhension du processus, etc.)

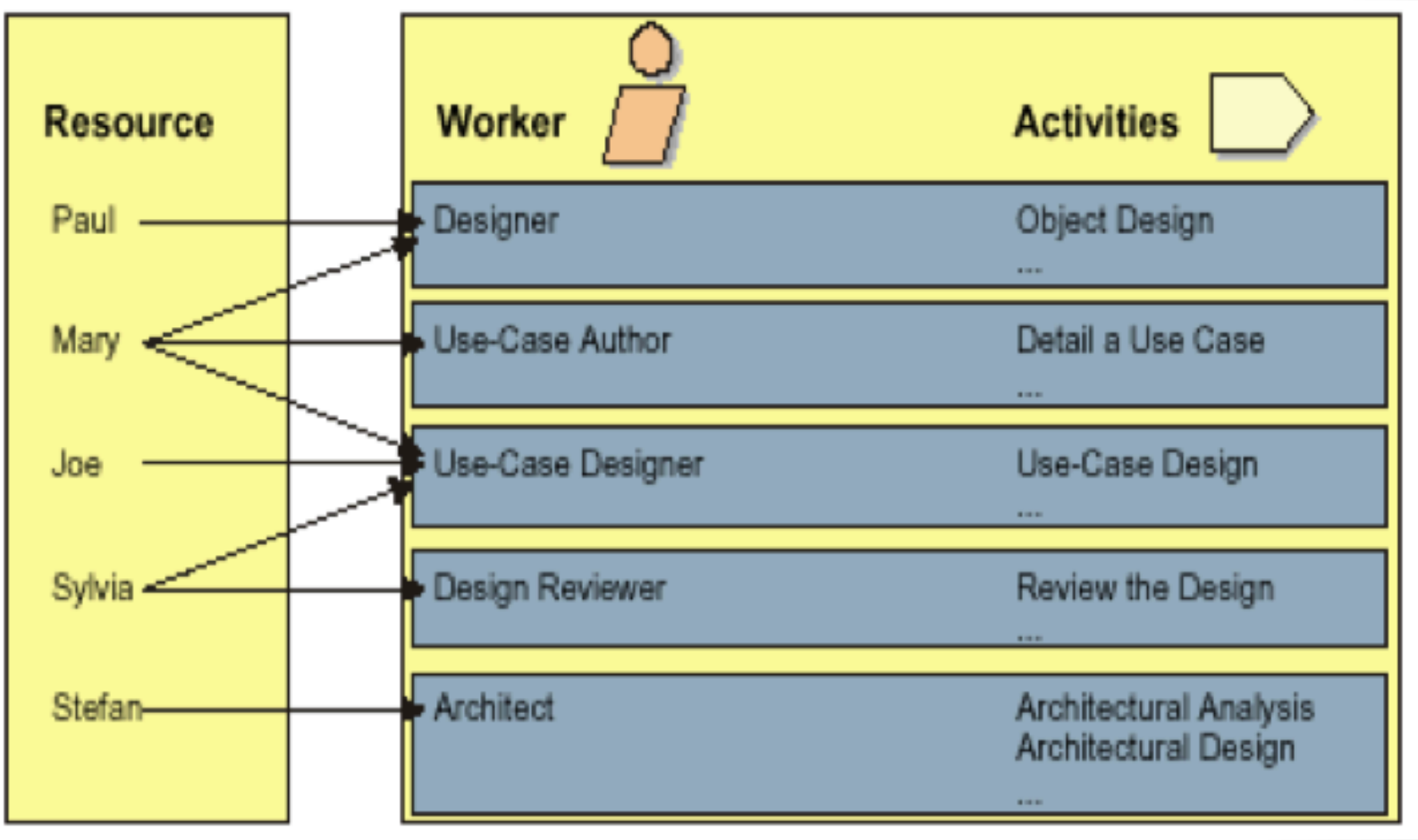
Activités, travailleurs, artefacts

- 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

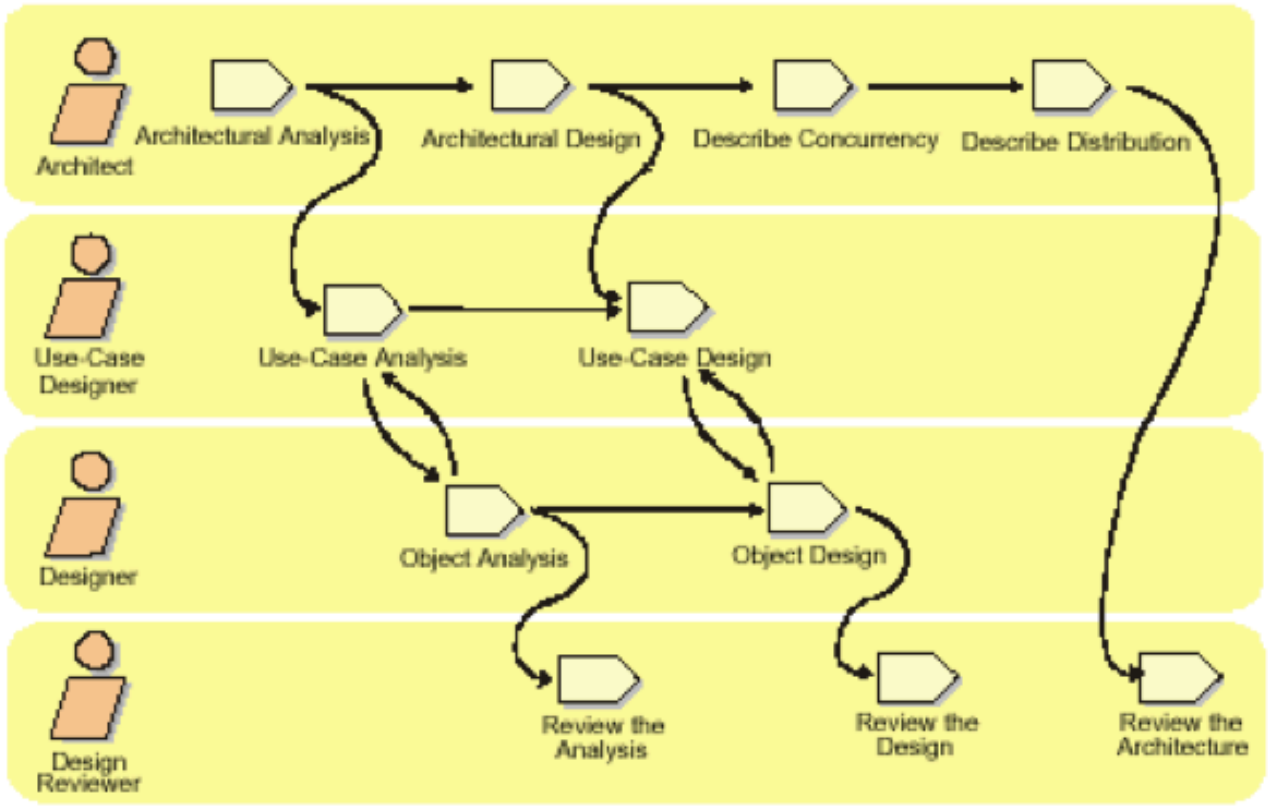
- Activités (comment)
  - 5 grandes activités, multiples sous-activités
    - tâches réalisée par un travailleur, impliquant une manipulation d'information
  - ex. : concevoir une classe, corriger un document, détailler un CU…

Travailleurs et activités



Workflows

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

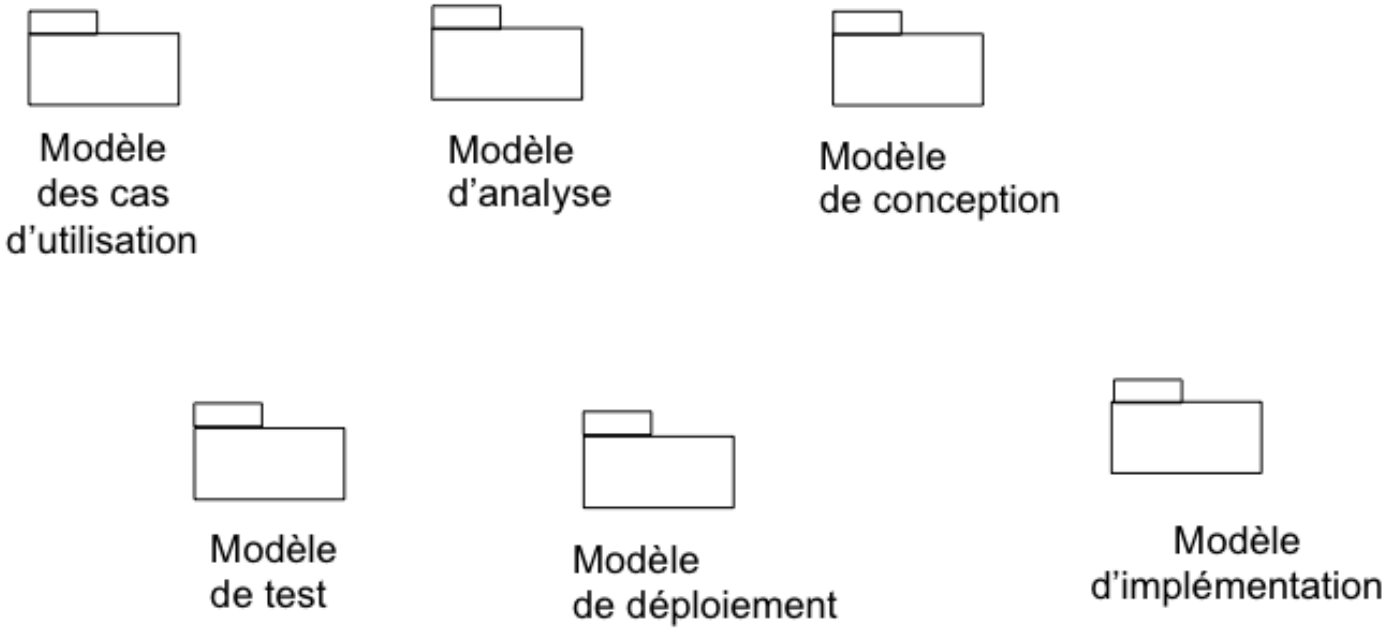


Les modèles du processus

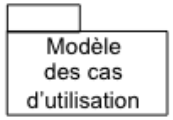
- Objectif
  - créer un modèle global composé de diagrammes liés aux différentes activités
    - et pas une montagne de documents
- Les modèles sont liés
  - par les CU
  - par les enchaînement d'activité qui les mettre en place
- Rappel
  - La description de l'architecture utilise une sous-partie des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

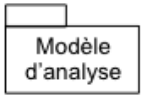
Modèles d'un projet USDP

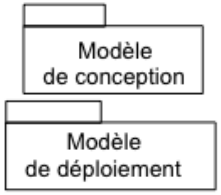
- Les activités consistent (entre autres) à créer des modèles

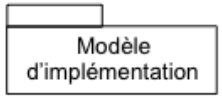


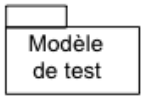
Modèles et activités











- 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

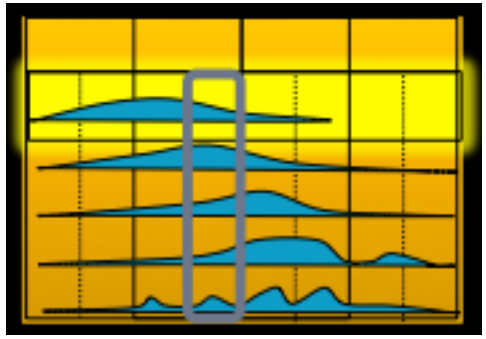
Plan

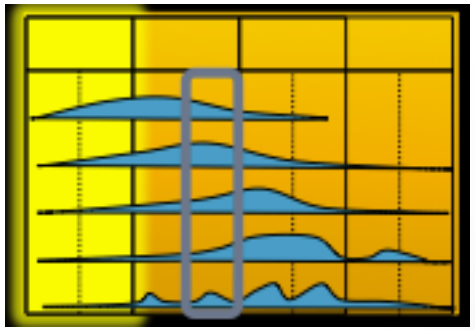
- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
    - Activités, travailleurs, artefacts & modèles
    - Différentes activités pour passer des besoins au code
    - Différentes phases pour piloter les activités
    - Quelques remarques
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Illustration du discours

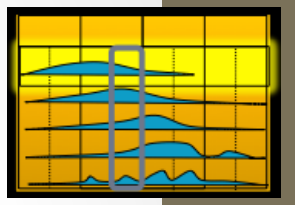
- Pour illustrer les différentes activités
  - Illustration avec la conception d'un SI pour la gestion d'un hôtel
  - Exemple tiré du cours de Yannick Prié
    - Lui même s'appuyant sur le cours de JL Sourrouile, et d'autres sources…

- Fil conducteur :





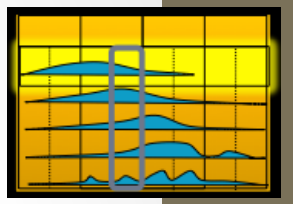
Activité : acquisition des besoins



Objectifs

- Déterminer les valeurs 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

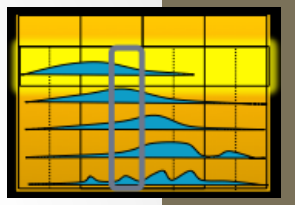
Activité : acquisition des besoins



Liste initiale des besoins

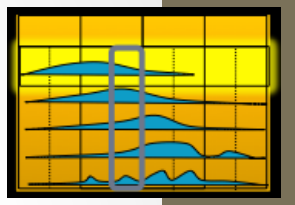
- Une petite chaîne d'hôtels a décidé de mettre sur le web un système de réservation de chambres ouvert à tout client, et d'autre part elle veut automatiser la gestion de ses hôtels.
- Un hôtel est géré par un directeur assisté d'employés.
- Pour réserver à distance, après avoir choisi hôtels et dates, le client fournit un numéro de carte bleue. Lorsque le retrait a été accepté, 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

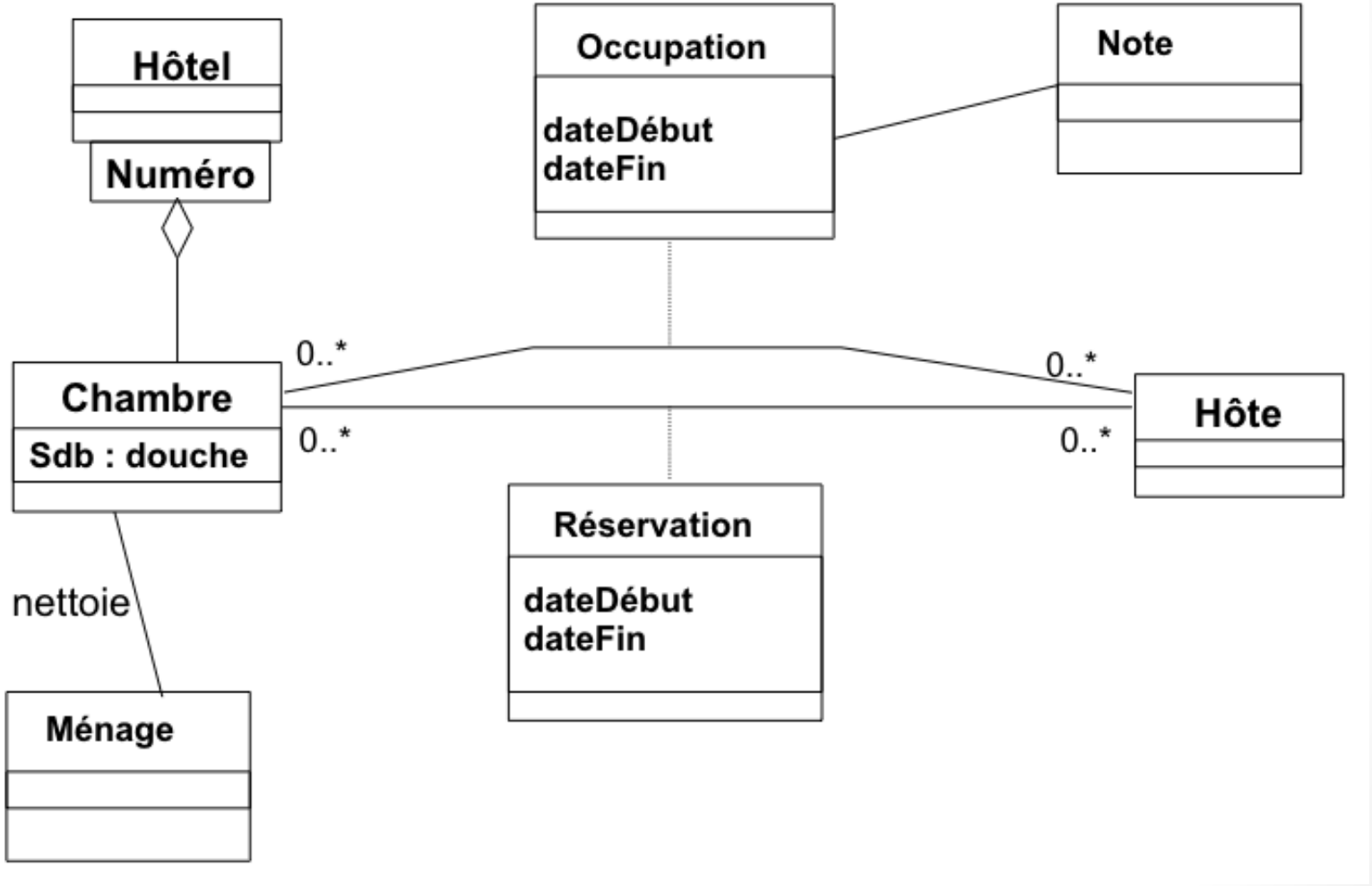
Activité : acquisition des besoins *Modèle du domaine*



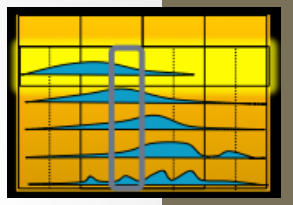
- Représentation des classes conceptuelles d'une situation réelle
  - objets métier (ex. Commande), du monde réel (ex. Avion), événements
  - quelques attributs, peu d'opérations
  - associations (s'il y a nécessité de conserver la mémoire de la relation)
  - les classes non retenues sont placées dans un glossaire
- « Dictionnaire visuel » du domaine construit surtout pendant la phase d'élaboration, itérativement en fonction des CU considérés

Activité : acquisition des besoins
Modèle du domaine



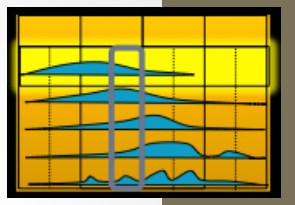


Activité : acquisition des besoins *Pour construire un modèle du domaine*

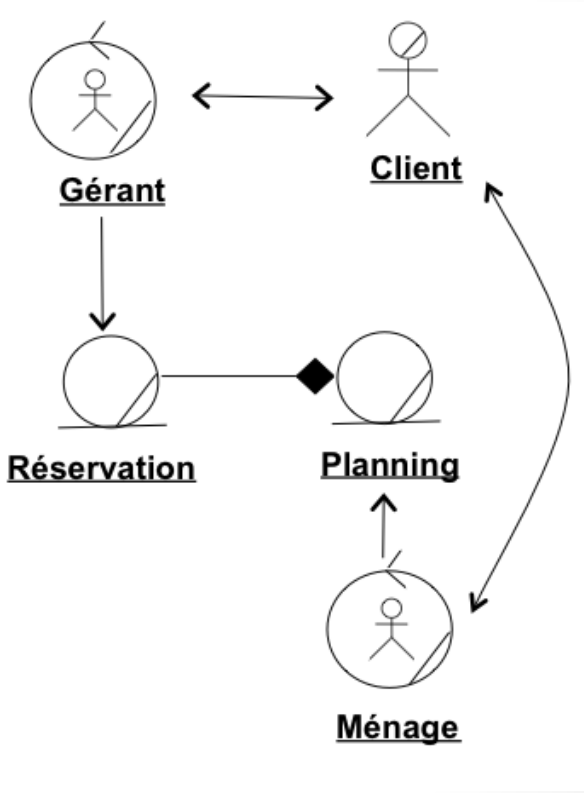


- Réutiliser/modifier des modèles existants
- Lister les catégories
  - classes : objets physiques, transactions, autre systèmes informatiques, organisations, documents de travail, etc.
  - associations : A est membre de B, A est une transaction liée à une transaction B, A est une description de B, etc.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

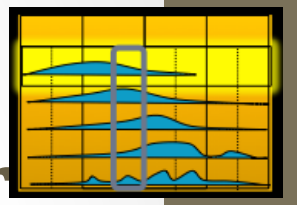
Activité : acquisition des besoins
Modèle du métier (facultatif)

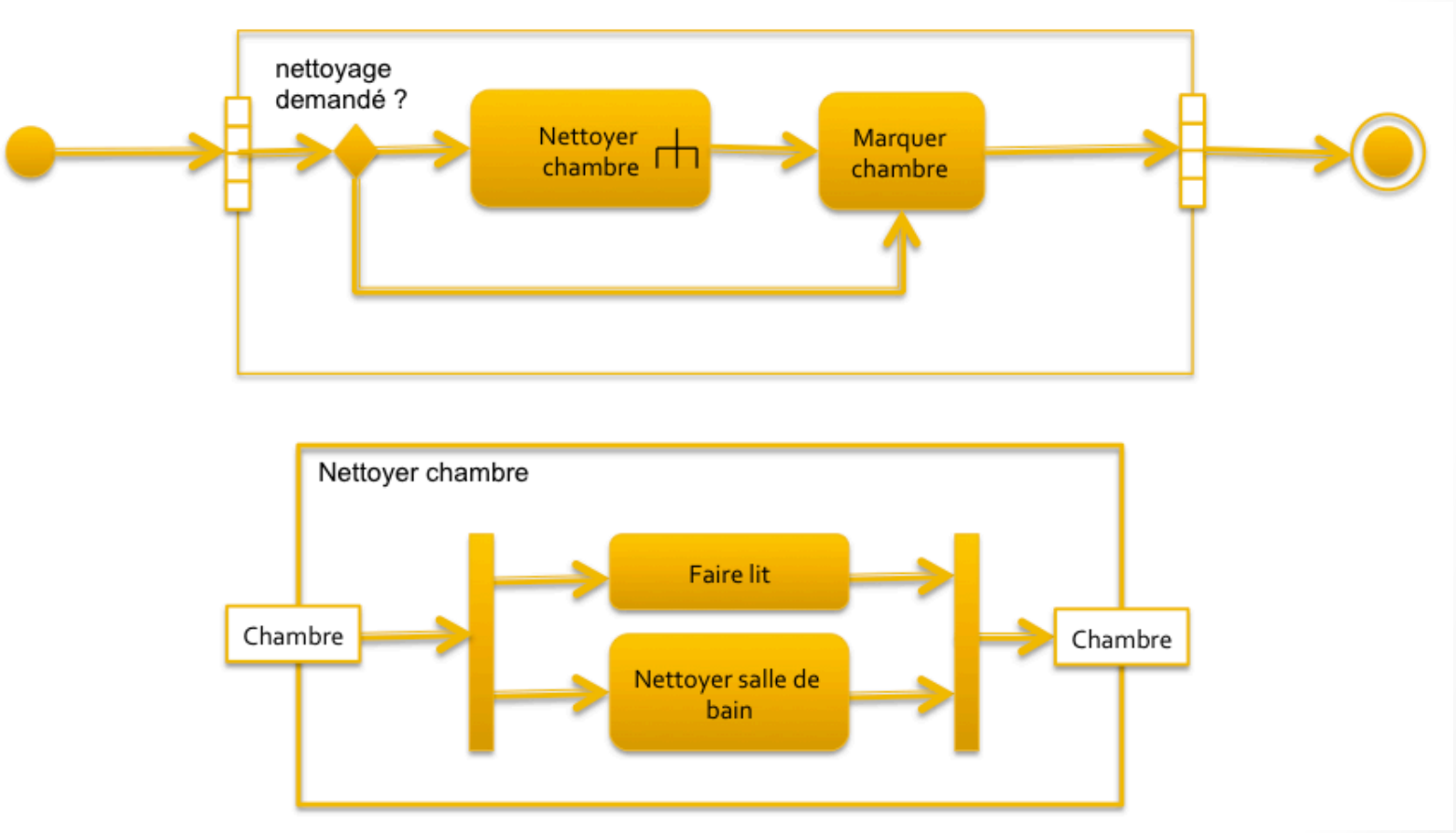


- Modèle des CU métier
  - représenter les processus
  - acteurs métier
- Modèle objet métier
  - montrer comment les CU métier sont réalisés par
    - acteurs métier
    - travailleurs métier
    - entités métier

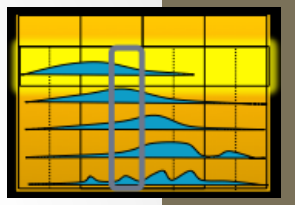


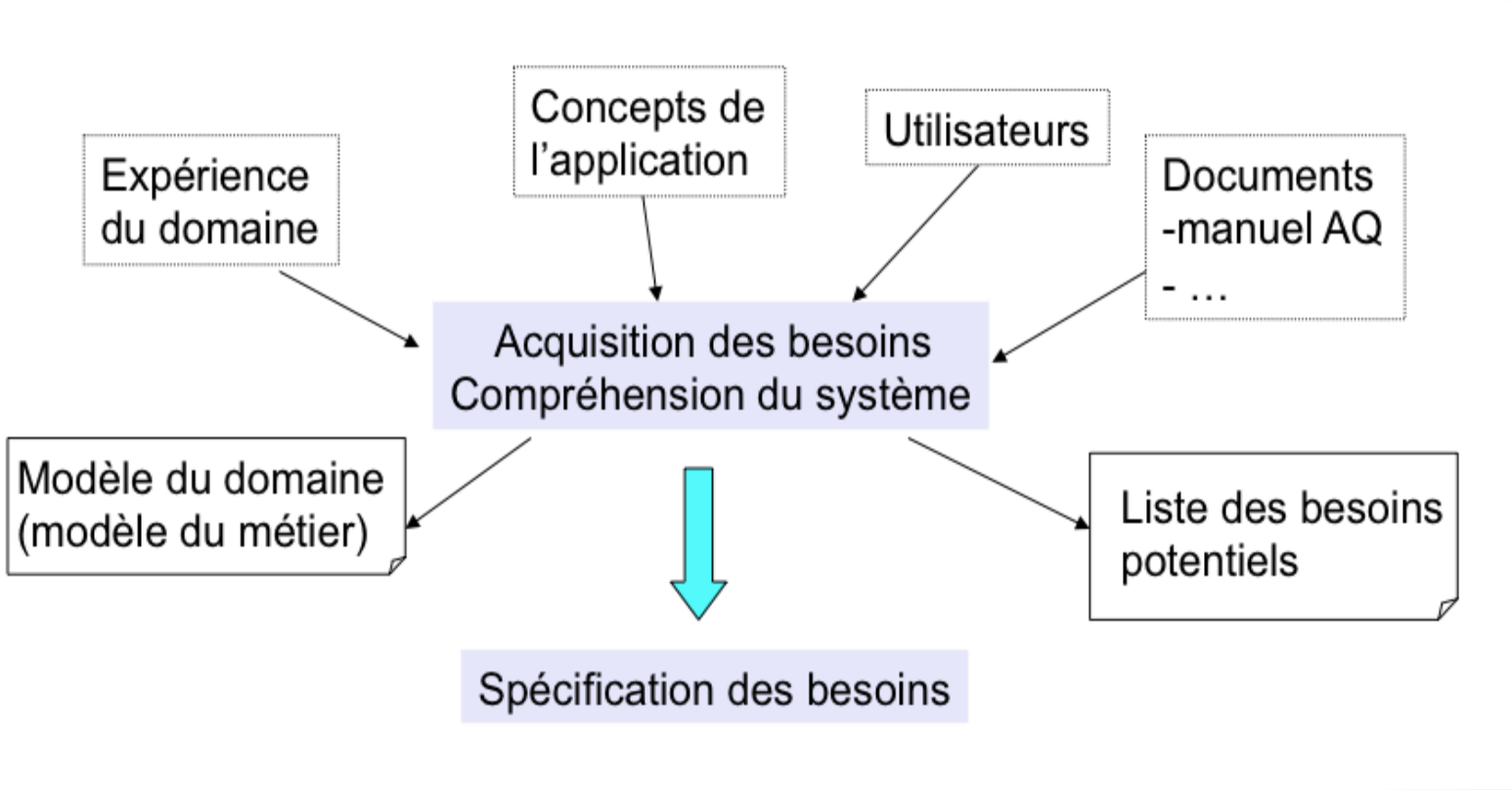
Activité : acquisition des besoins
Modélisation des processus métier





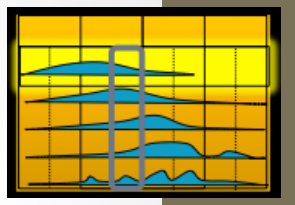
Activité : acquisition des besoins
Produits de l'acquisition des besoins





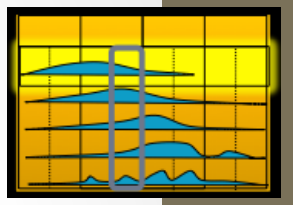
Activité : expression des besoins

Objectifs

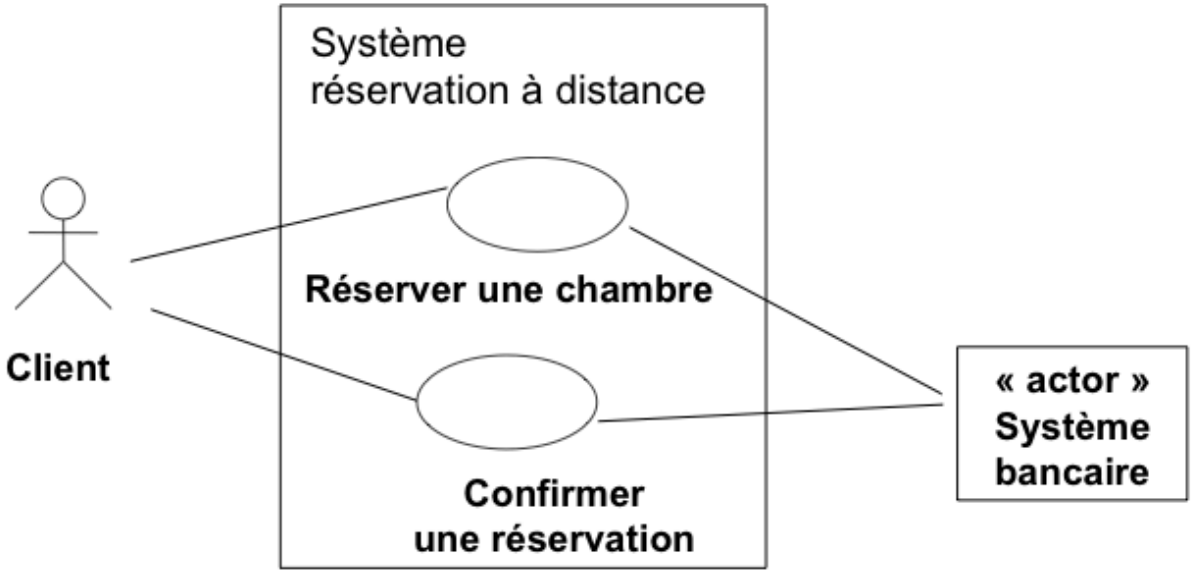


1. Définir les besoins fonctionnels
   - Écrire les récits d'utilisation
   - Définir les acteurs et leurs objectifs
     - Limites du système à construire
       - Interactions avec le système
   - Définir les cas d'utilisation
     - Portée système / objectif utilisateur
     - Description brève, puis détaillée
   - Construire le modèle des cas d'utilisation
     - Organiser les cas d'utilisation
       - Utiliser les CU portés organisation

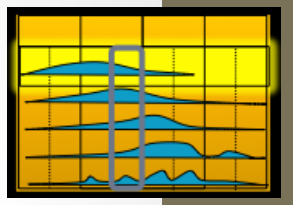
Activité : expression des besoins
Ex. description des CU

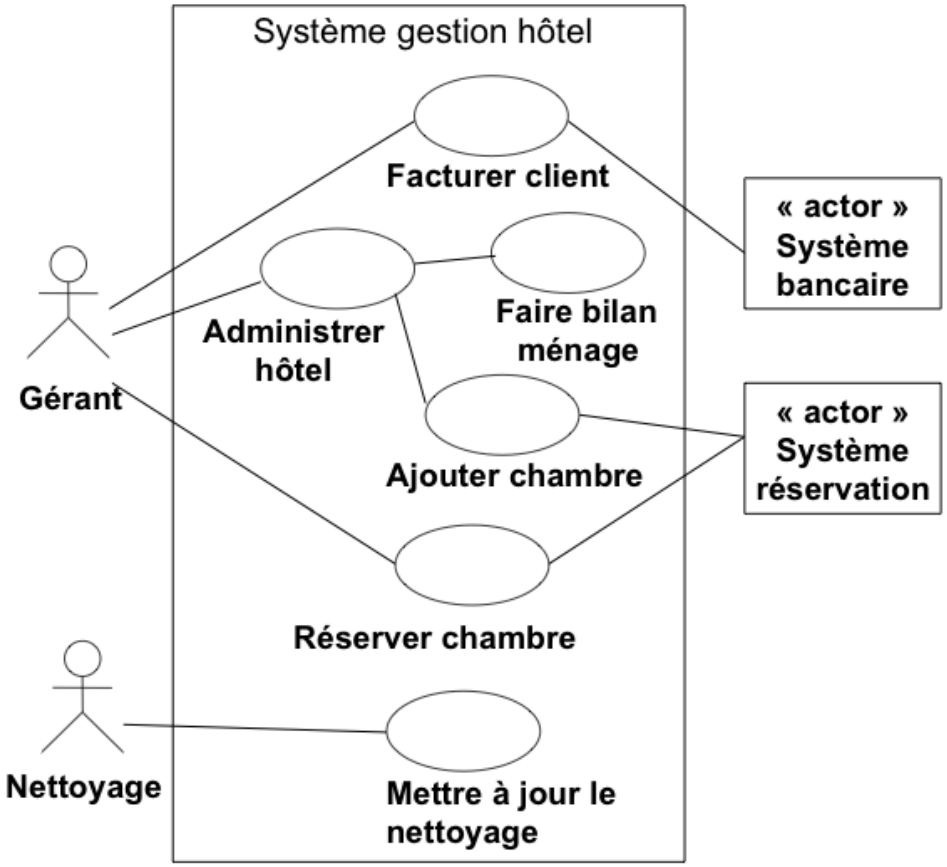


Le système à développer est divisé en deux sous-systèmes indépendants : le système de réservation à distance et le système de gestion local à l'hôtel.
La base de données des réservations est considérée comme un système externe mais non détaillé (sous-système classique).
Localement, la base de données est considérée comme interne au système et ignorée pour l'instant.



Activité : expression des besoins
Ex. description des CU





Administrer hôtel

Le gérant gère les 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...)...

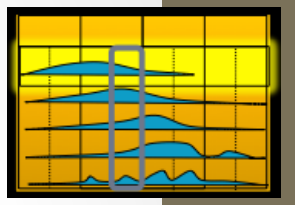
Ajouter chambre

Le gérant crée une chambre dans le système, avec toutes ses caractéristiques.

Facturer client

...

Activité : expression des besoins
Objectifs

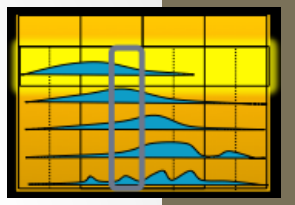


2. Classer les cas d'utilisation par priorité
   - la priorité dépend des risques associés au cas d'utilisation et de leur importance pour l'architecture, des nécessités de réalisation et de tests

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. Le système de gestion des chambres pour le ménage peut entraîner des questions techniques difficiles (équipement mobile).

1. Réserver une chambre sur le web
2. Réserver une chambre (locale)
3. Trouver la prochaine chambre à nettoyer
4. Administrer
5. ...

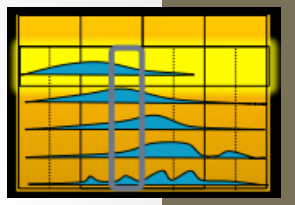
Activité : expression des besoins
Objectifs



3. Détailler et formaliser les cas d'utilisation
   - un cas d'utilisation représente un ensemble de scénarios
   - donner pour chaque cas les scénarios principaux
   - détailler le déroulement des scénarios (texte)
   - compléter la description des scénarios
     - diagrammes de séquences système, d'activité, d'états…
     - Maquette IHM, si l'interface est complexe ou nécessite une validation par le client
4. Structurer le modèle
   - Si besoin, révision des cas

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

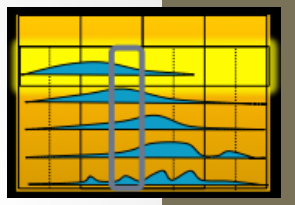
Activité : expression des besoins
Ex. Détails des CU



- Description détaillée
  - Scénario nominal, extension (cf. cours sur CU)

CU : Réserver 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.
…

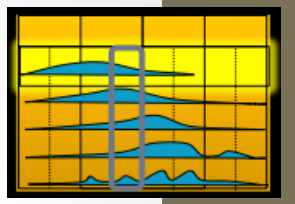
Activité : expression des besoins
Objectifs

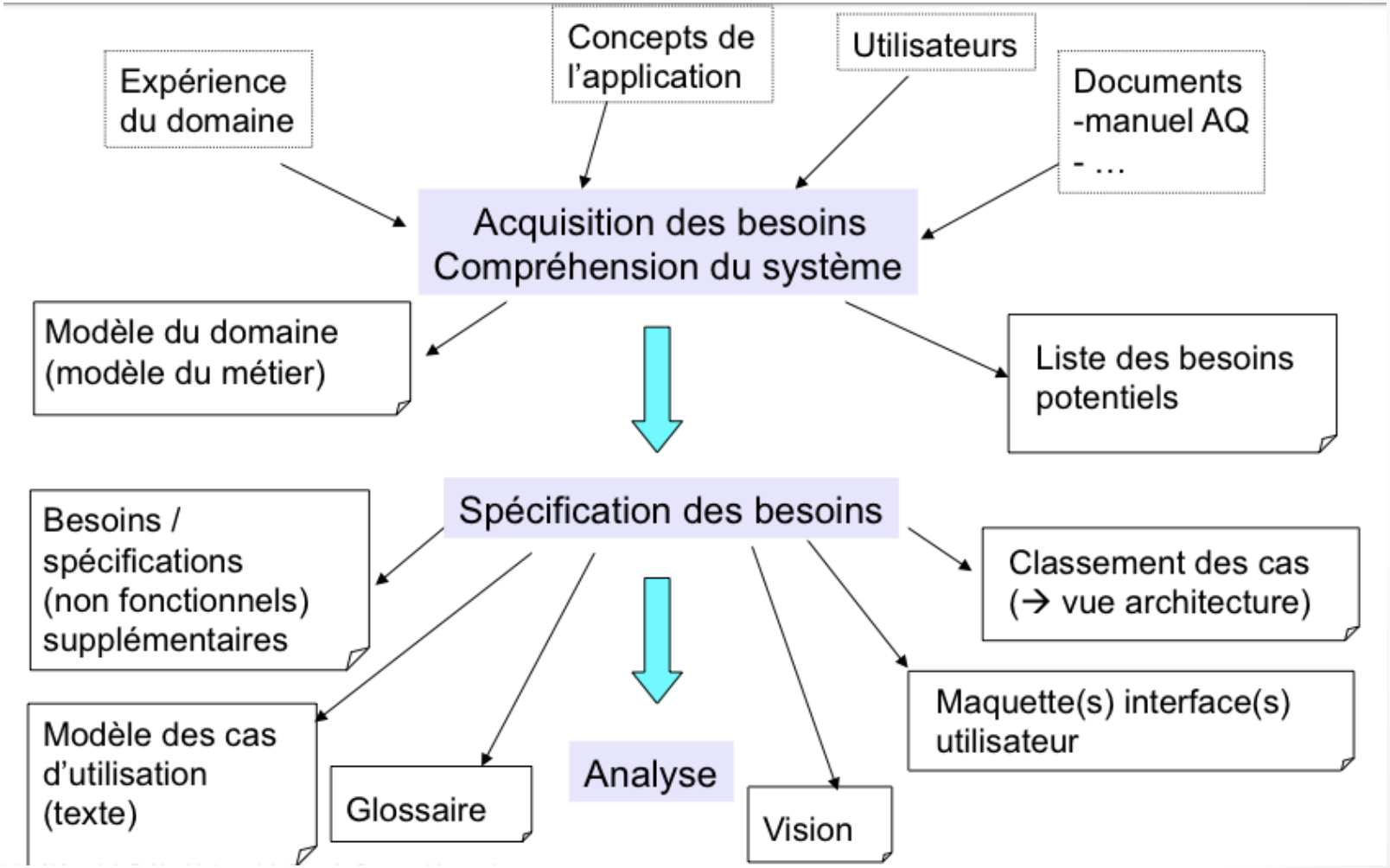


5. Appréhender les besoins non fonctionnels (contraintes sur le système : environnement, plate-forme, fiabilité, vitesse…)
   - rattacher si possible les besoins aux cas d'utilisation
     - description dans les descriptions des CU (section « exigences particulières »)
   - sinon, dresser une liste des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

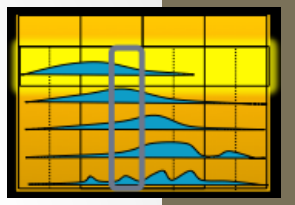
> La chaîne possède 117 hôtels de 30 chambres en moyenne. Les appels sur le réseau sont évalués à 300 par jour (au début, prévoir des évolutions).
>
> Pour des raisons d'extensibilité, de performances et de sécurité, la chaîne de traitement des réservations des clients doit être indépendante des liaisons des hôtels avec le système de réservation. Les hôtels sont reliés en permanence au système de réservation aux coupures près (ADSL) et les postes devront être fiables (reprise sur erreur après 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.

Activité : expression des besoins
Artefacts



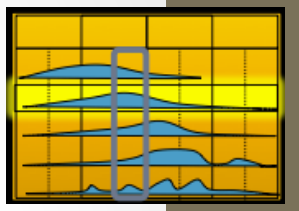


Activité : expression des besoins
Travailleurs



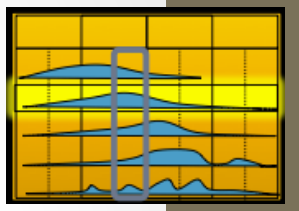
- Analyste système, du domaine
  - modèle du domaine / du métier
  - modèle des CU / acteurs
  - Glossaire
- Spécificateur de cas d'utilisation
  - CU détaillés
- Concepteur d'interface utilisateur
  - maquette/prototype
- Architecte
  - vue architecturale du modèle des CU

Activité : analyse
Objectif principal



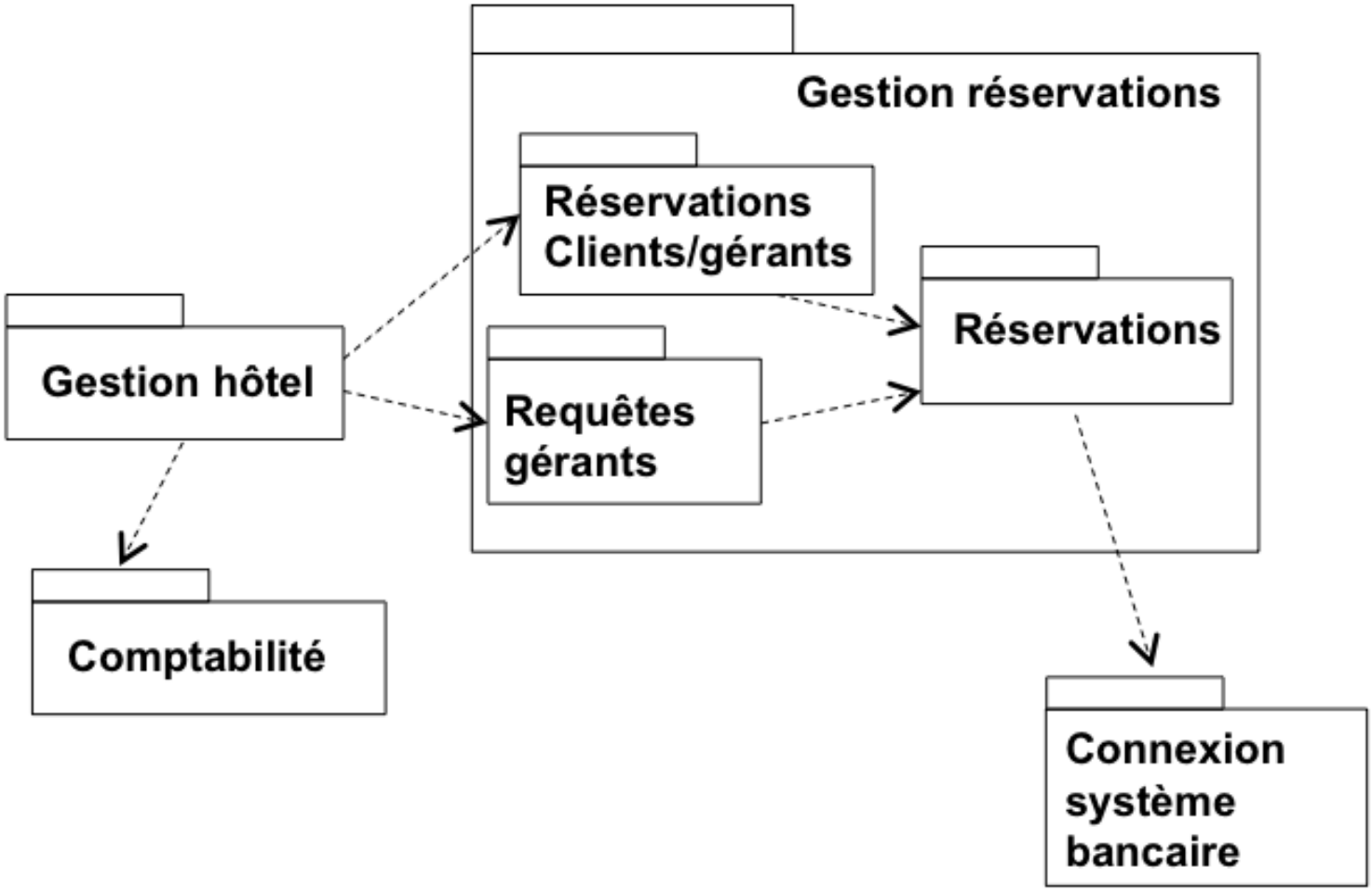
- Construire un modèle d'analyse pour préparer la conception
  - forme générale stable du système
  - haut-niveau d'abstraction
  - vision plus précise et formelle des CU, réalisation par des objets d'analyse
  - passage du langage du client à celui du développeur

Activité : analyse
Objectifs

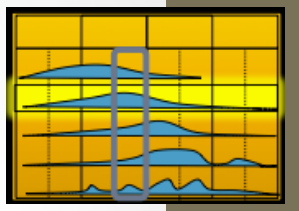


1. Mener l'analyse architecturale
   - identifier les paquetages d'analyse
     - regroupement logique indépendant de la réalisation
     - relations de dépendances, navigabilité entre classes de paquetage différents
     - à partir des CU et du domaine
     - servira de point de départ pour le découpage en sous-systèmes

Activité : analyse
Découpage en paquetages

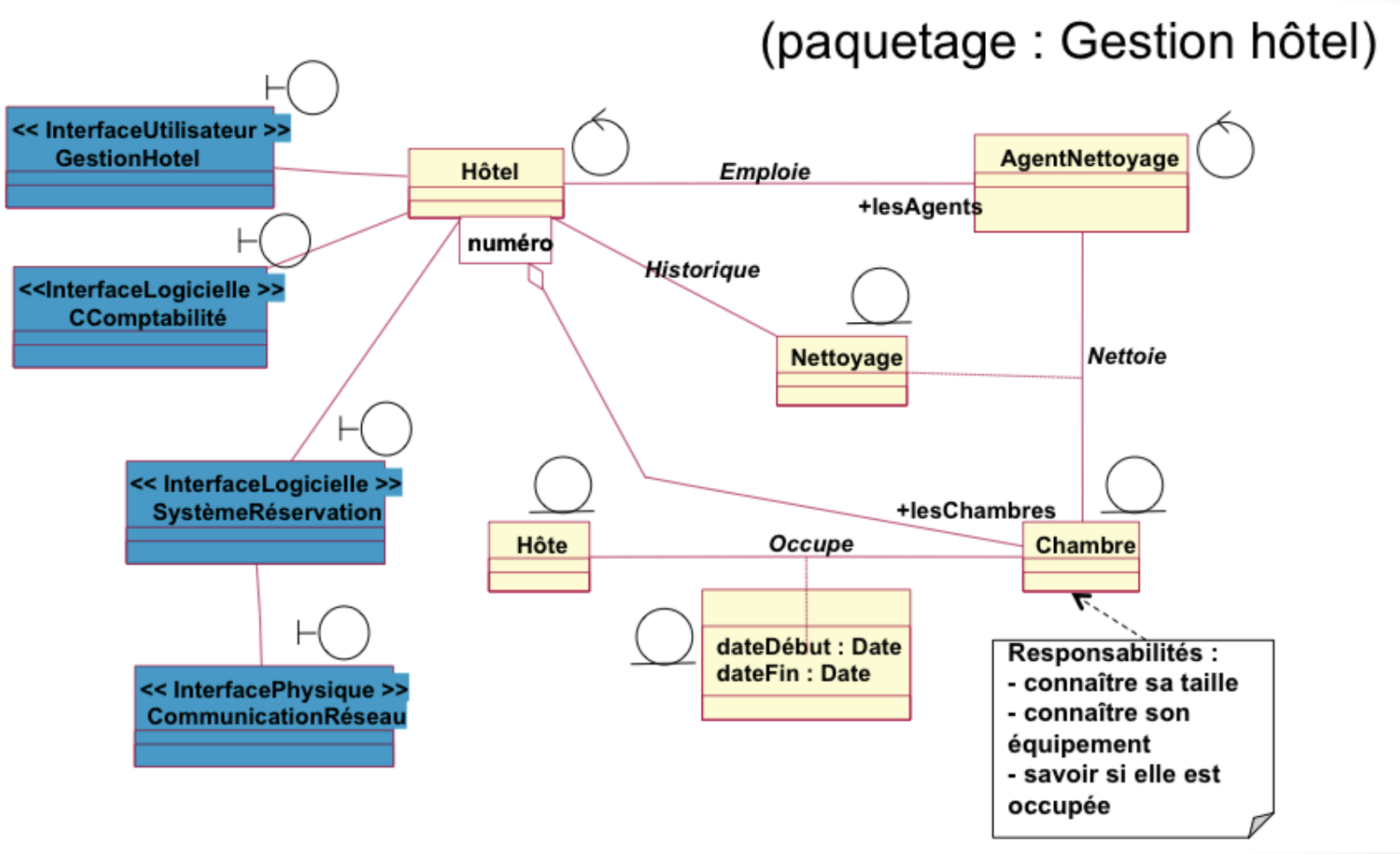


Activité : analyse
Objectifs

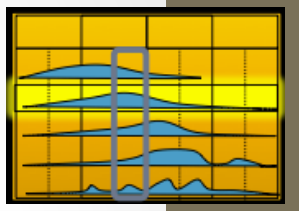


1. Mener l'analyse architecturale (suite)
   - Identifier les classes entités manifestes (premier modèle structurel)
     - Modèle des 10-20 classes constituant l'essence du domaine (à partir du modèle domaine/métier)
     - Responsabilités évidentes
   - Identifier les exigences particulières communes
     - Distribution, sécurité, persistance, tolérance aux fautes…
     - Les rattacher aux classes et cas d'utilisation

Activité : analyse
Premier modèle structurel



Activité : analyse
Objectifs

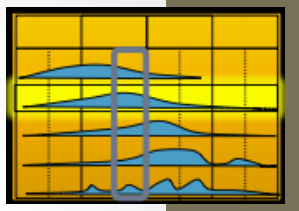


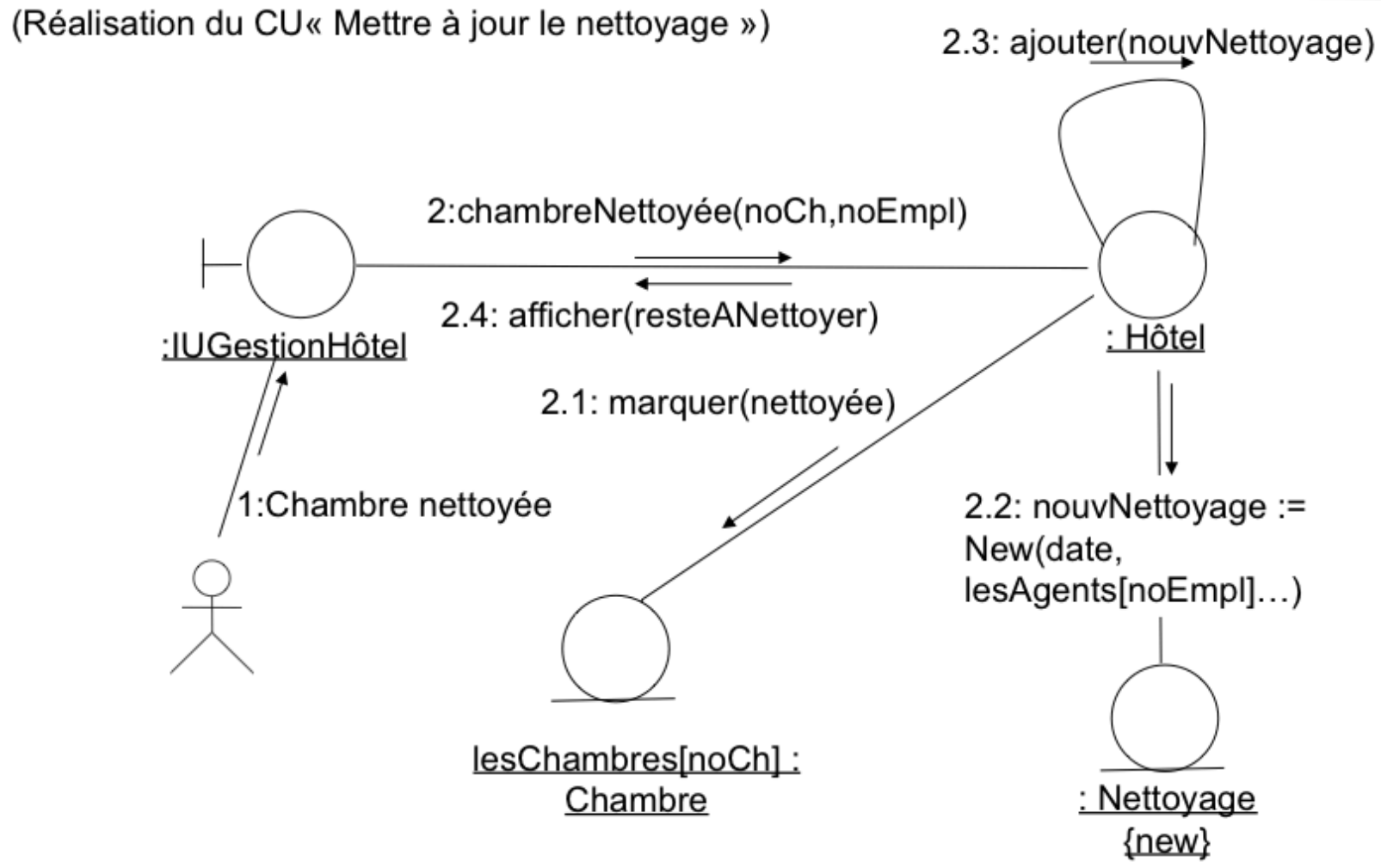
2. Analyser les cas d'utilisation
   - raffiner tous les scénarios des cas d'utilisation

   ⇒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
     - raffiner les diagrammes de séquence système des scénarios
     - construire les diagrammes de collaboration
     - si scénario trop complexe, modéliser par parties (enchaînements), et indiquer les branchements

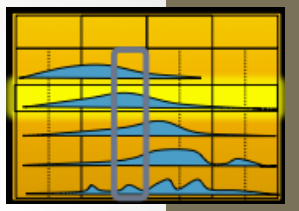
Le modèle structurel sera construit pour supporter l'union des collaborations et interactions exprimées

Activité : analyse
Diagramme de communication



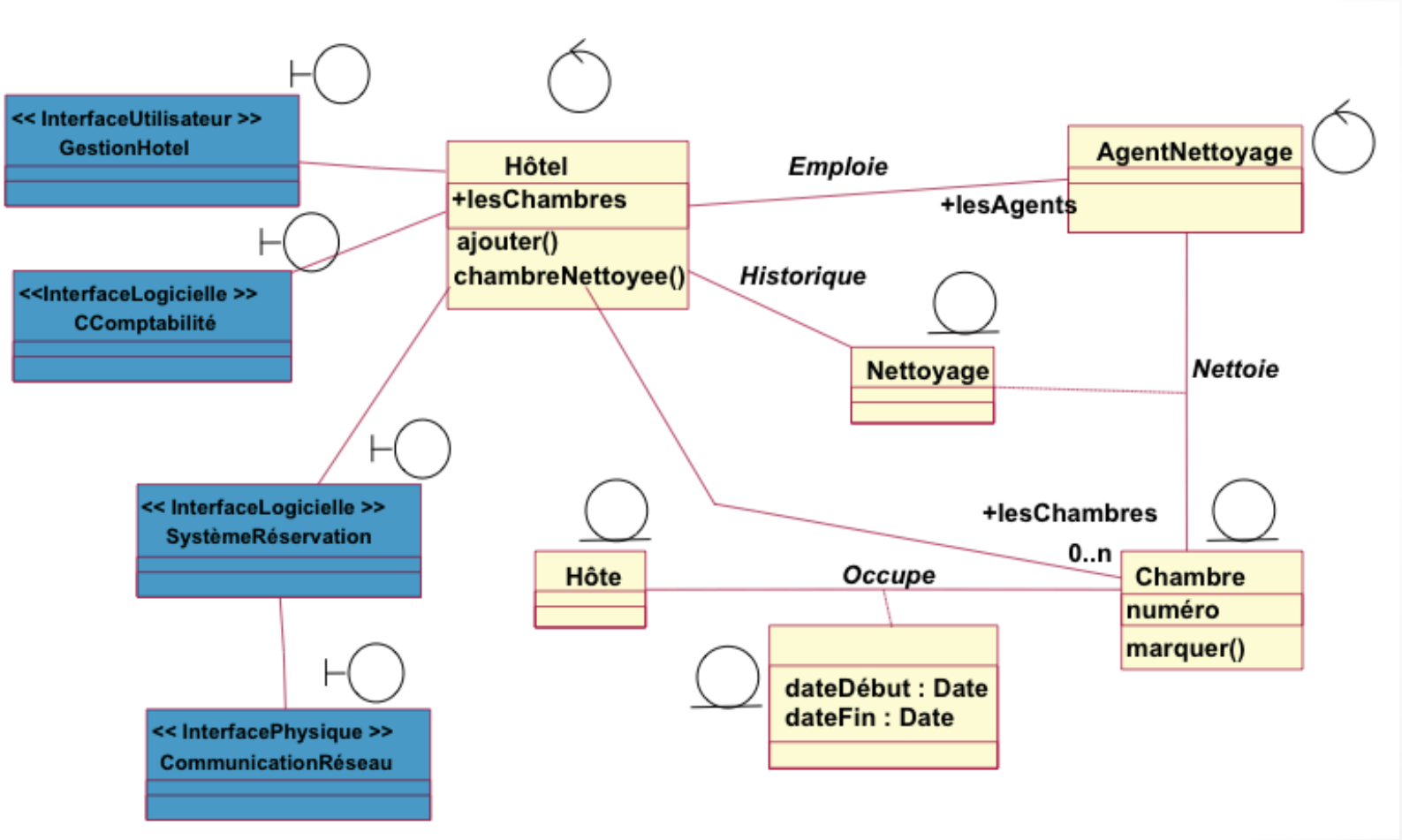


Activité : analyse
Objectifs

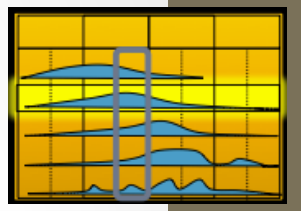


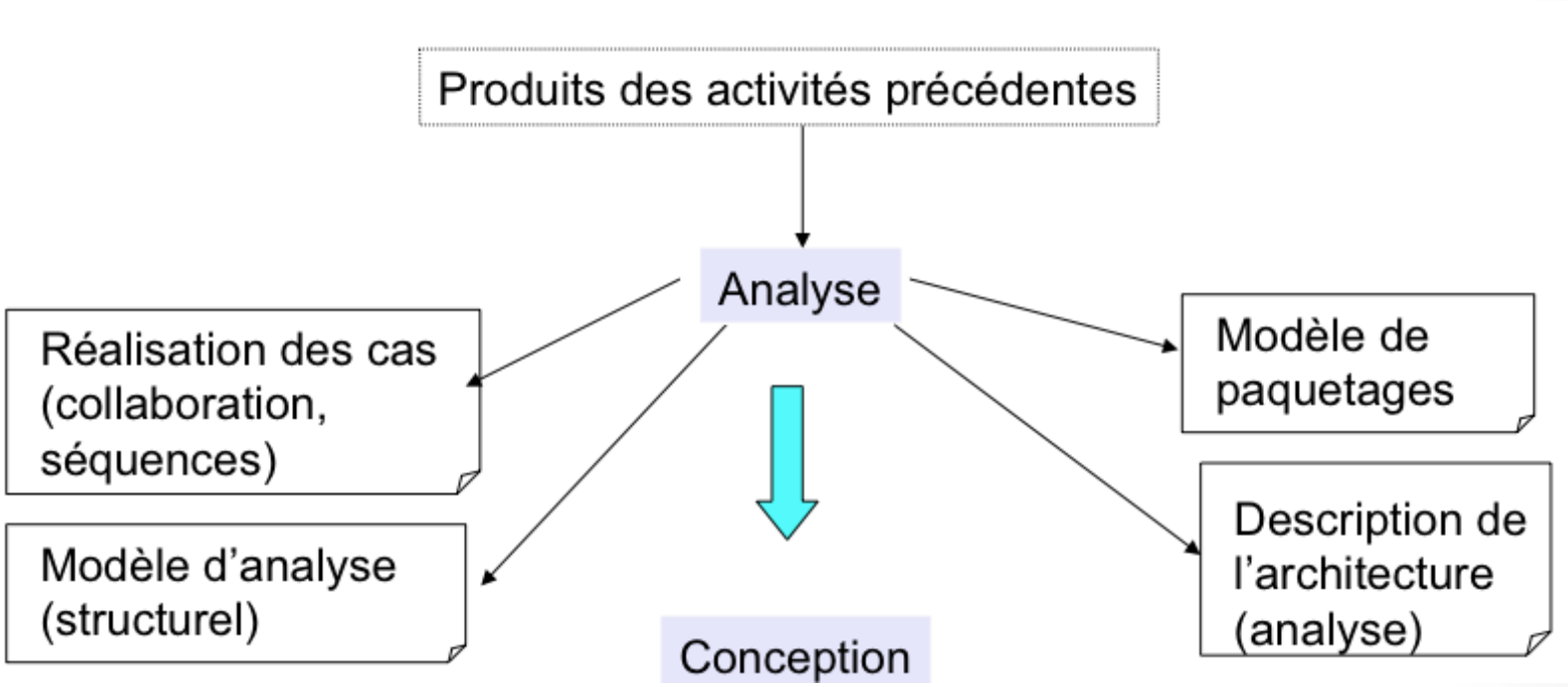
3. Préciser les classes d'analyse
   - faire le bilan des responsabilités à partir des collaborations
     - responsabilité d'une classe = union des rôles dans tous les cas (négliger en analyse les opérations implicites)
   - identifier les attributs
     - rester simple, pas choix de conception à ce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

Activité : analyse
Diagramme de classe complété

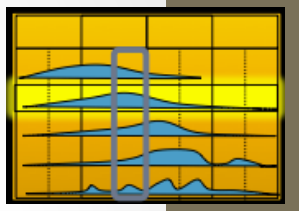


Activité : analyse
Produits de l'analyse



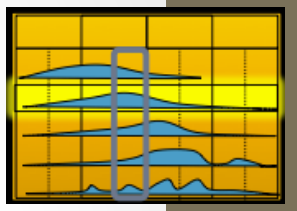


Activité : analyse
Travailleurs



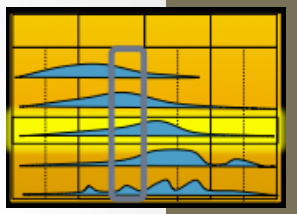
- Architecte
  - intégrité du modèle d'analyse
  - description de l'architecture
    - extrait du modèle d'analyse
- Ingénieur des CU
  - réalisation/analyse des CU
- Ingénieur des composants
  - classes d'analyse
  - paquetages d'analyse

Activité : analyse
Comparaison des modèles CU et analyse



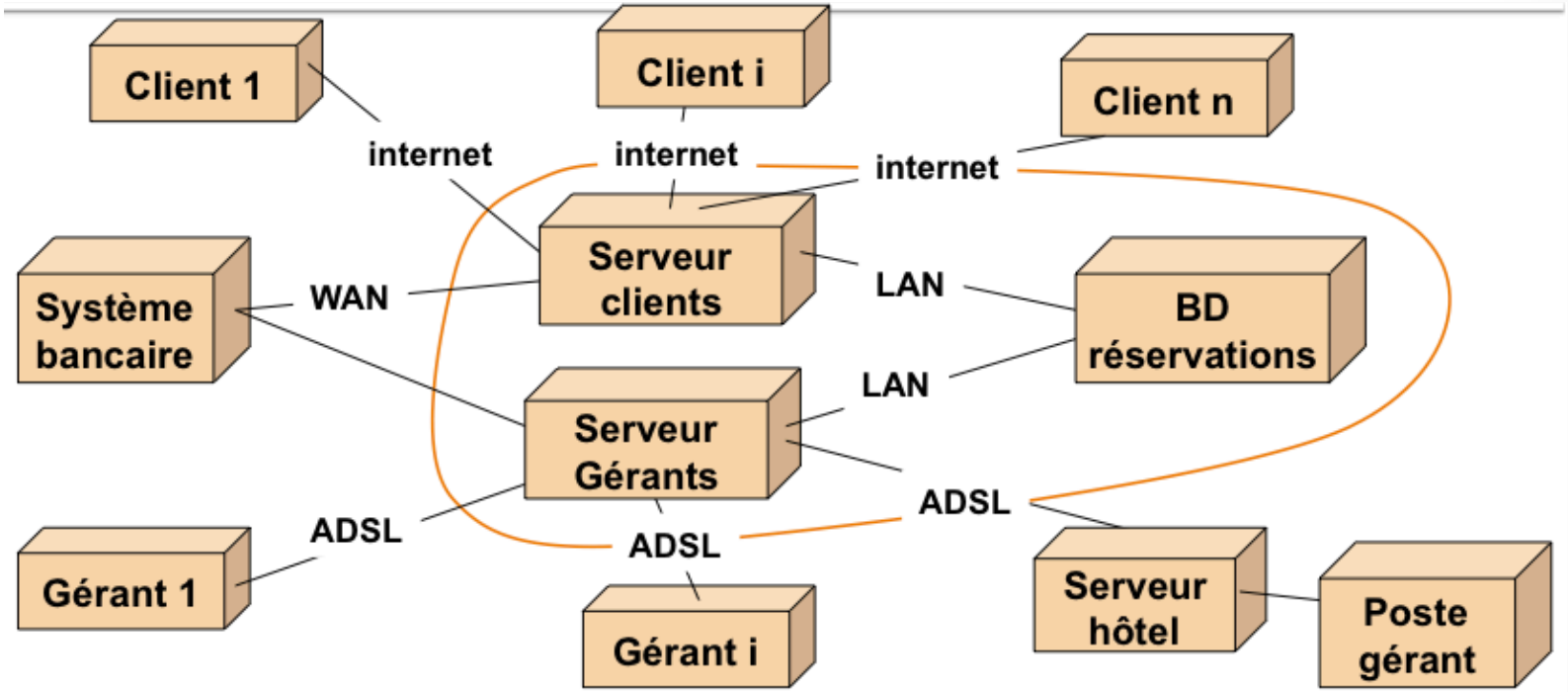
Modèle des cas d'utilisation	Modèle d'analyse
Langage du client	Langage du développeur
Vue externe du système	Vue interne du système
Structuré par les cas (vue externe)	Structuré par les classes et paquetages (vue interne)
Utilisé comme "contrat" avec le client	Utilisé pour comprendre le système
Informel	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

Activité : conception
Objectifs



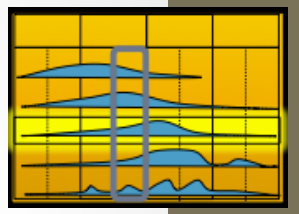
- Proposer une réalisation de l'analyse et des cas d'utilisation en prenant en compte toutes les exigences
- Mener la conception architecturale
  - identifier les nœuds et la configuration du réseau (déploiement),
  - les sous-systèmes et leurs interfaces (modèle en couche en général),
  - les classes 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...
  - composants de service liés à l'appli, de middleware…
  - Permettra de distribuer le travail aux développeurs

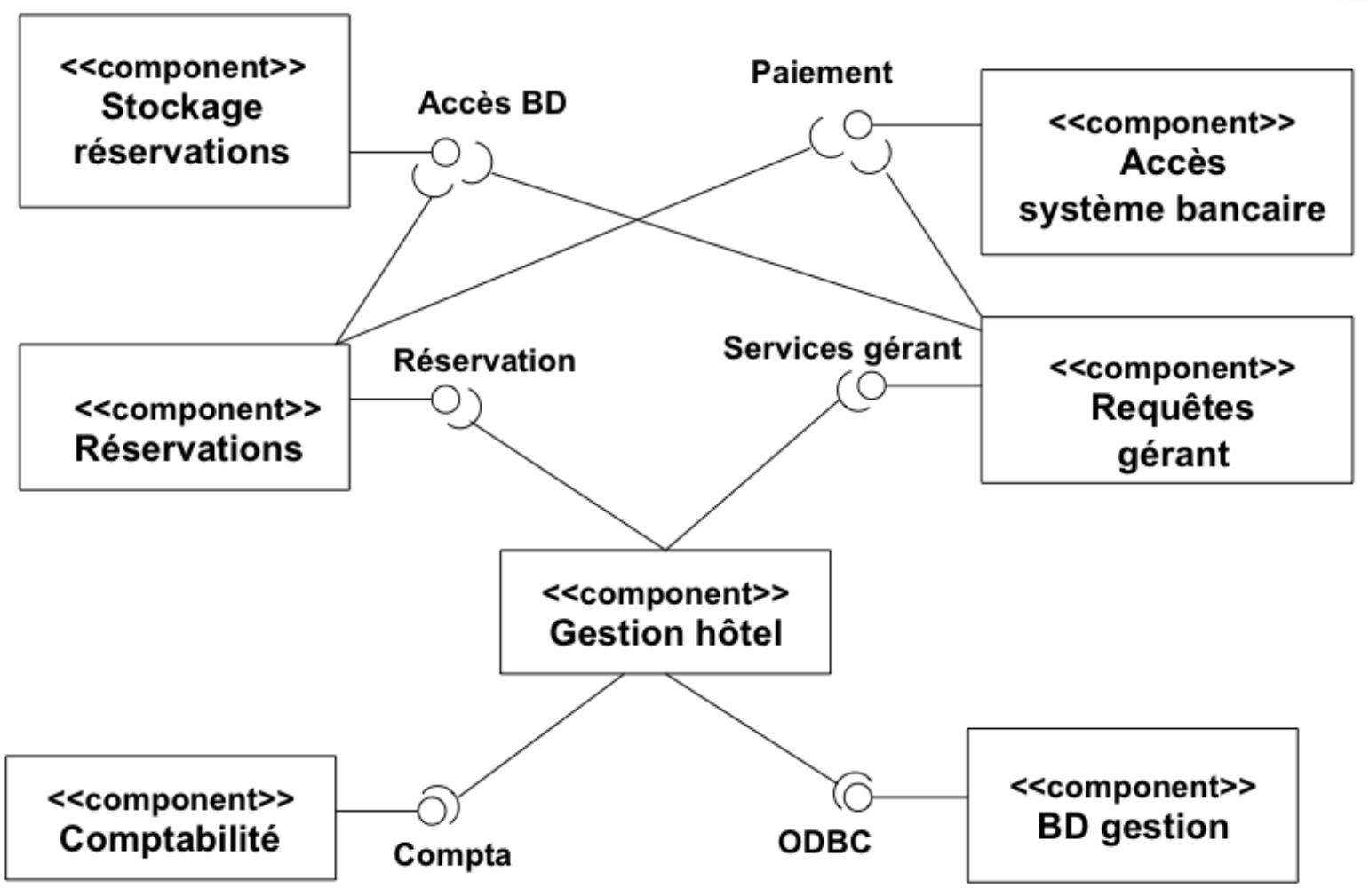
Activité : conception
Diagramme de déploiement



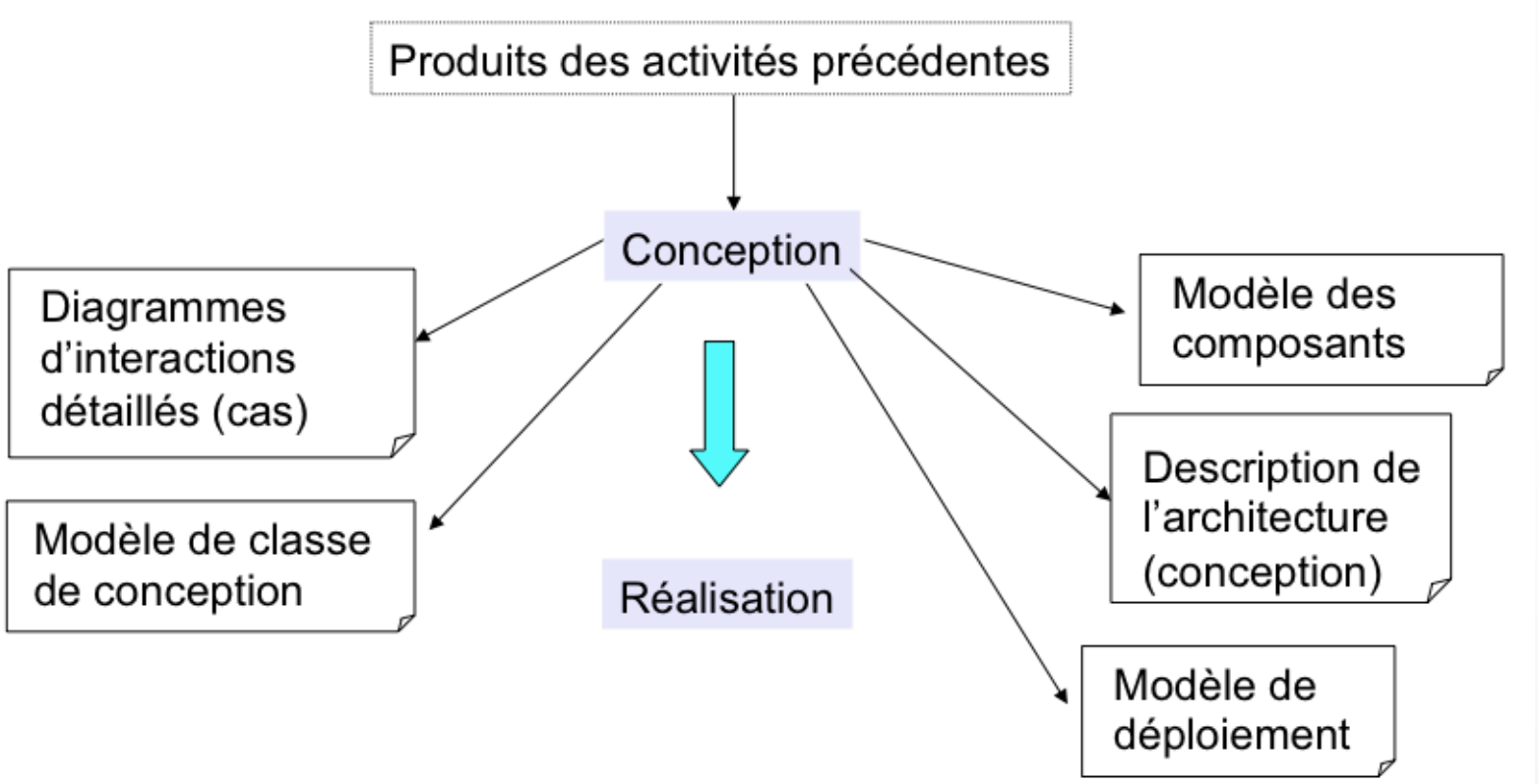
- Les deux serveurs et la BD des réservations peuvent être sur un même nœud tant que les liaisons clients ne pénalisent pas les liaisons gérants
- Les liaisons gérant-serveur sont en ADSL
- Possibilité que les hôtel aient un unique poste, ou bien un serveur local
- ...

Activité : conception
Diagramme de déploiement

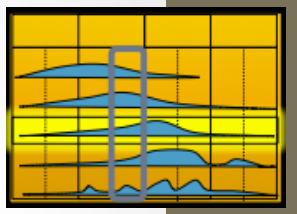




Activité : conception
Produits de la conception

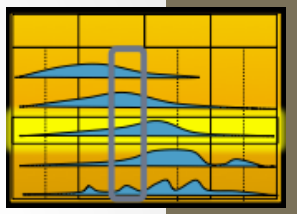


Activité : conception
Travailleurs



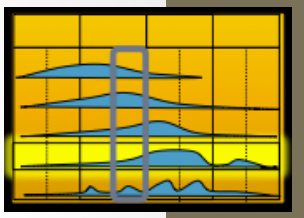
- Architecte
  - intégrité des modèles de conception et de déploiement
  - description de l'architecture
- Ingénieur de CU
  - réalisation/conception des CU
- Ingénieur de composants
  - classes de conception
  - sous-systèmes de conception
  - interfaces

Activité : conception
Comparaison des modèles analyse et conception



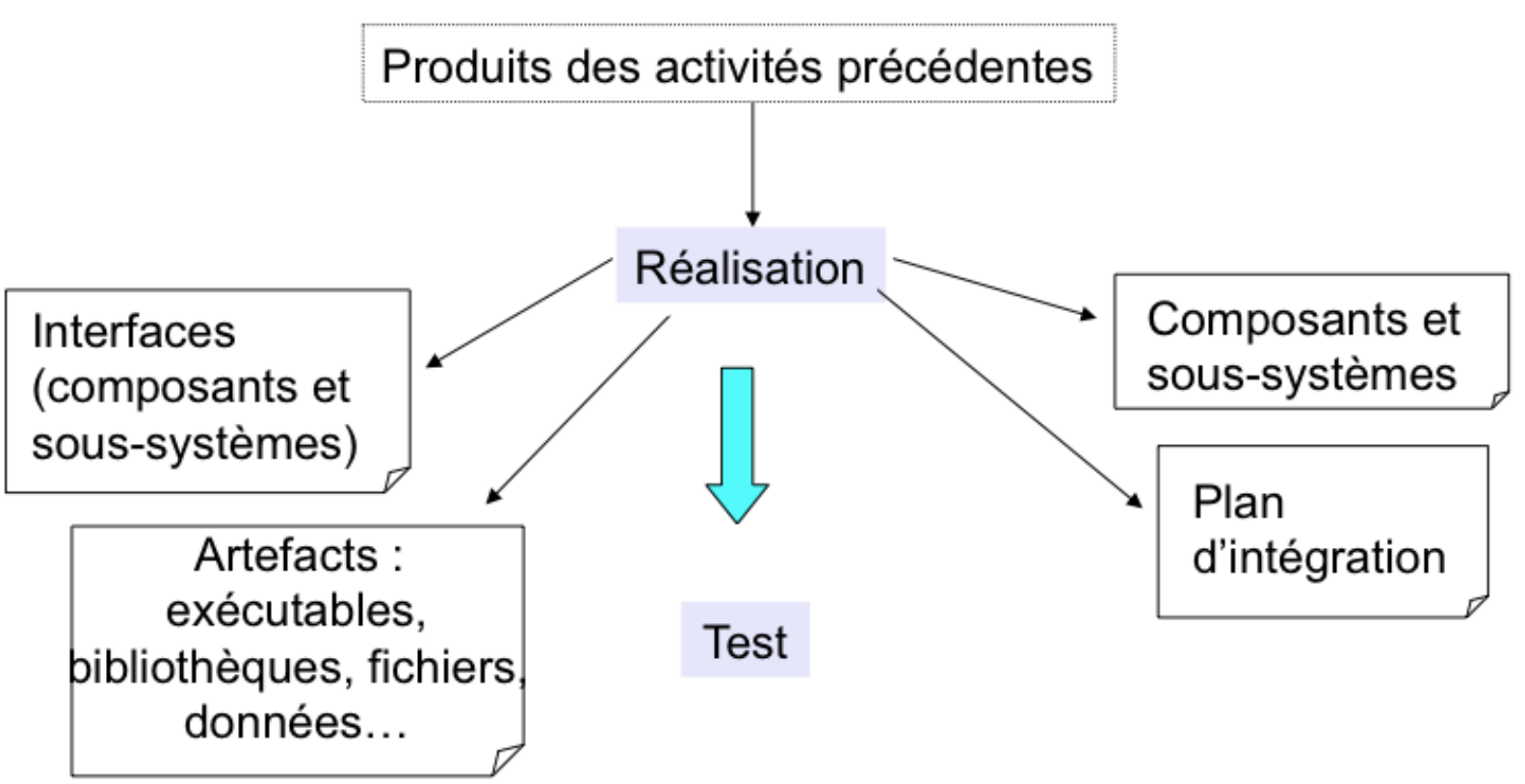
Modèle d'analyse	Modèle de conception
Modèle conceptuel, abstraction du système	Modèle physique qui sera mis en oeuvre
Générique vis à vis de la conception	Spécifique
Moins formel	Plus formel
Donne les grandes lignes de la conception, dont l'architecture. Définit une structure essentielle pour le système	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 (UML mode plan)

Activité : réalisation
Objectifs

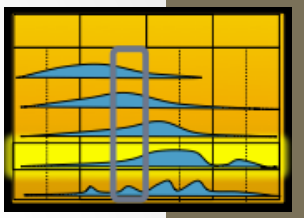


- Faire la mise en œuvre architecturale
  - identifier les artefacts logiciels et les associer à des nœuds
- Intégrer le système
  - planifier l'intégration, intégrer les incréments réalisés
- Réaliser les composants / sous-systèmes
- Réaliser les classes
- Mener les tests unitaires

Activité : réalisation
Produits de la réalisation

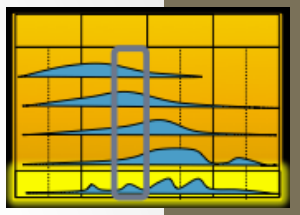


Activité : réalisation
Travailleurs



- Architecte
  - modèles d'implémentation 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

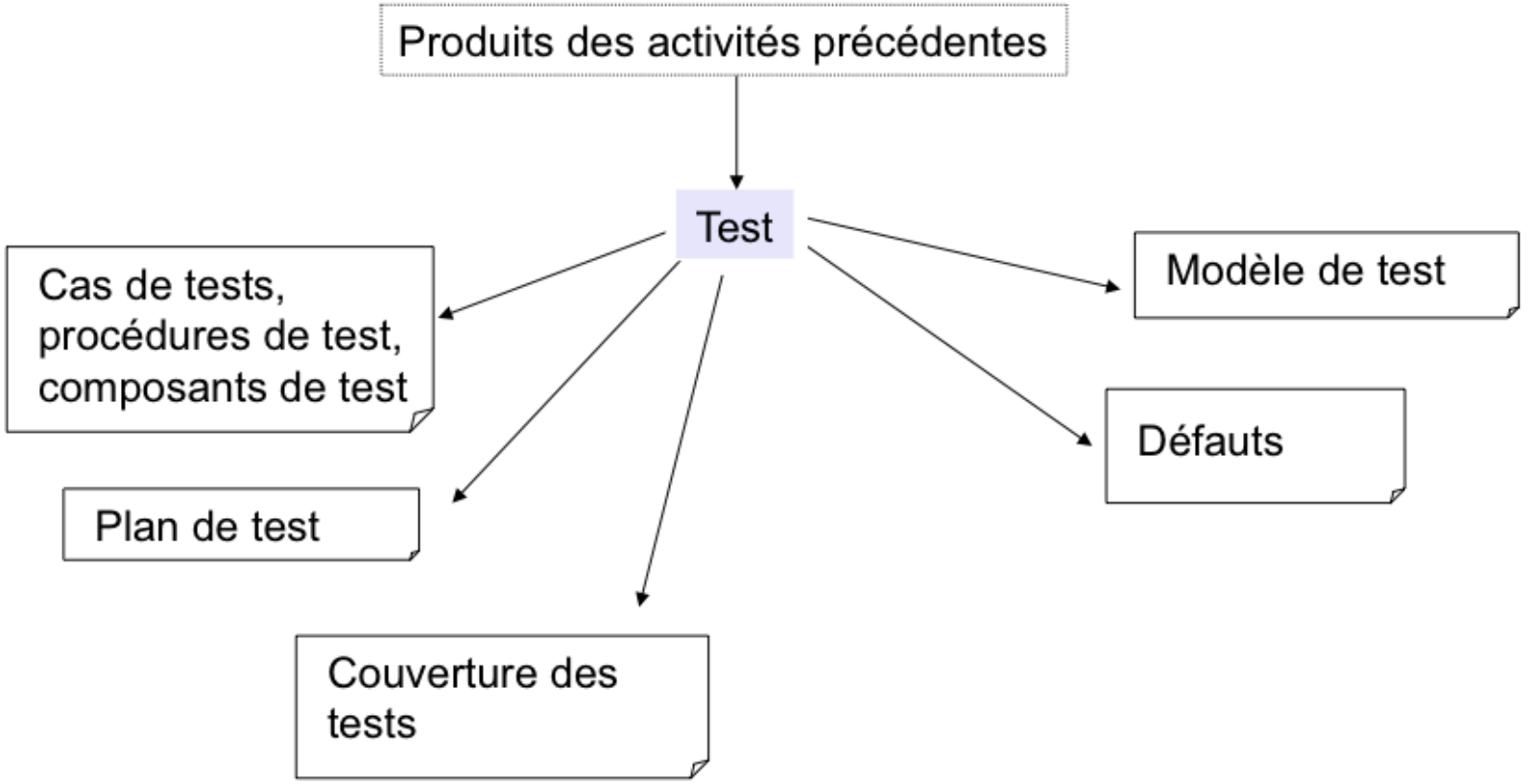
Activité : test
Objectifs



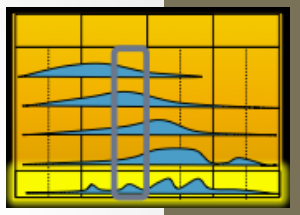
- Rédiger le plan de test
  - décrire la stratégie de test
  - estimer les besoins pour l'effort de test
  - planifier l'effort dans le temps

- Concevoir les tests
- Automatiser les tests
- Réaliser les tests d'intégration
- Réaliser les tests du système
- Évaluer les tests

Activité : test
Produits de la réalisation



Activité : test
Travailleurs

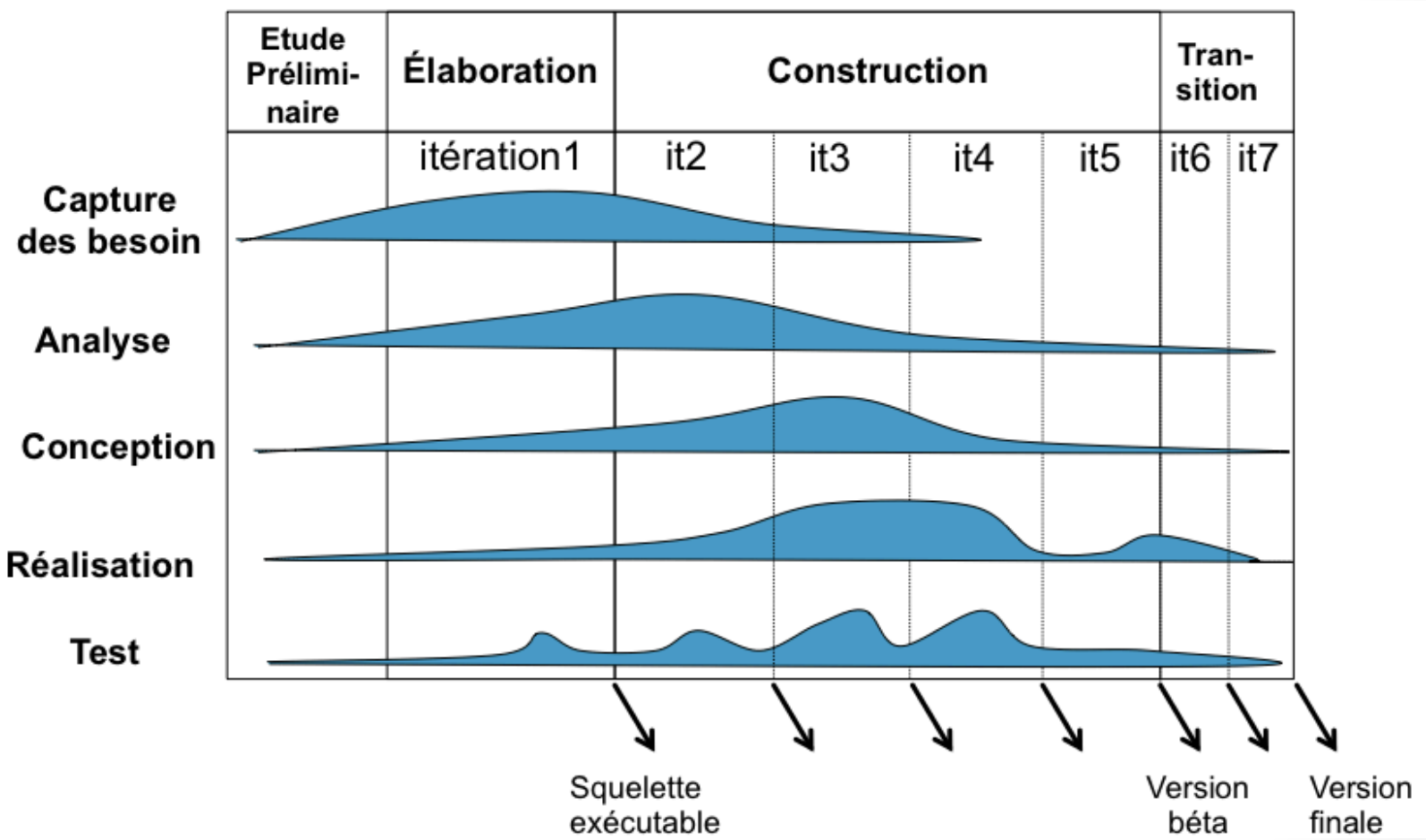


- Concepteur de tests
  - modèle de tests
  - cas de test
  - procédures de test
  - évaluation des tests
  - plan de tests
- Ingénieur de composants
  - test unitaires
- Testeur d'intégration
  - tests d'intégration
- Testeur système
  - vérification du système dans son ensemble

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
    - Activités, travailleurs, artefacts & modèles
    - Différentes activités pour passer des besoins au code
    - Différentes phases pour piloter les activités
    - Quelques remarques
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Un exemple de projet à 7 itérations



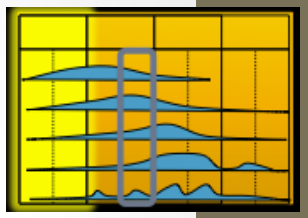
Gestion des phases

- 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

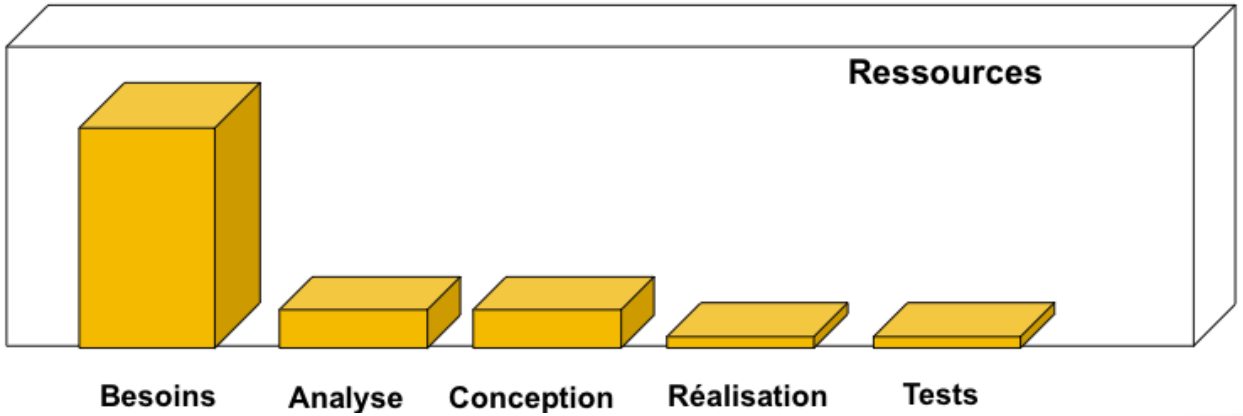
- Dans chaque phase
  - planifier les itérations et leurs objectifs de manière à réduire
    - les risques spécifiques du produit
    - les risques de ne pas découvrir l'architecture adaptée
    - les risques de ne pas satisfaire les besoins
  - définir les critères d'évaluation de fin d'itération

- Dans chaque itération
  - faire les ajustements indispensables (planning, modèles, processus, outils...)

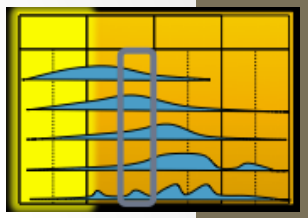
Phase d'étude préliminaire



- 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

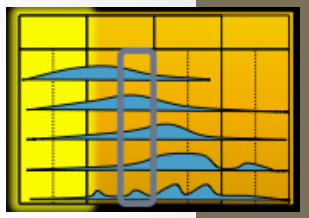


- Capture des besoins
  - comprendre le contexte du système (50-70% du contexte)
  - établir les besoins fonctionnels et non fonctionnels (80%)
  - traduire les besoins fonctionnels en cas d'utilisation (50%)
  - détailler les premiers cas par ordre de priorité (10% max)

- Analyse
  - analyse des cas d'utilisation (10% considérés, 5% raffinés)
  - pour mieux comprendre le système à réaliser, guider le choix de l'architecture

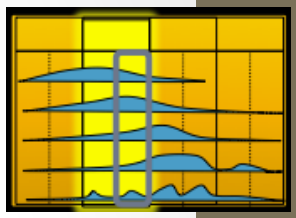
- Conception
  - première ébauche de la conception architecturale : sous-systèmes, nœuds, réseau, couches logicielles
  - examen des aspects importants et à plus haut risque

Livrables de l'étude préliminaire

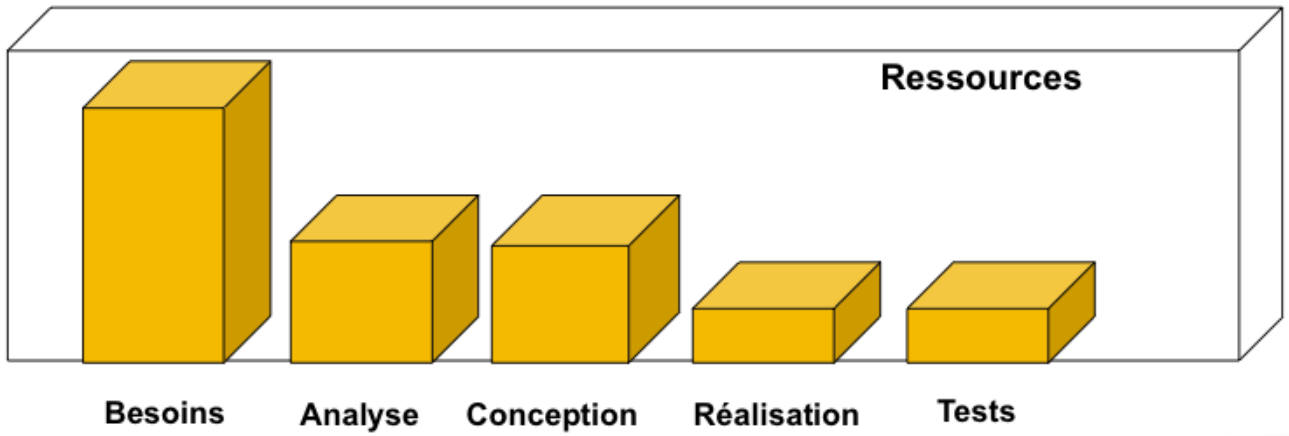


- Première version du modèle du domaine ou de contexte de l'entreprise
  - parties prenantes, utilisateurs
- Liste des besoins
  - fonctionnels et non fonctionnels
- Ébauche des modèles de cas, d'analyse et de conception
- Esquisse d'une architecture
- Liste ordonnée de risques et liste ordonnée de cas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

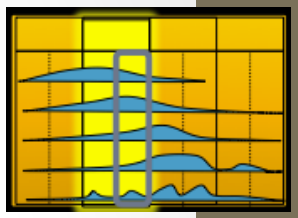
Phase d'élaboration



- 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

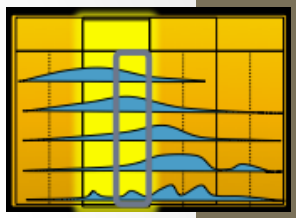


Activités principales de l'élaboration



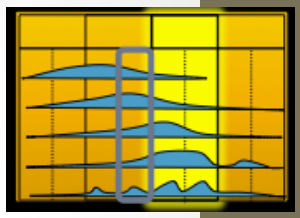
- Capture des besoins
  - terminer la capture des besoins et en détailler de 40 à 80%
  - faire un prototype de l'interface utilisateur (éventuellement)
- Analyse
  - analyse architecturale complète (paquetages...)
  - raffinement des cas d'utilisation (pour l'architecture, < 10%)
- Conception
  - terminer 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'étude préliminaire

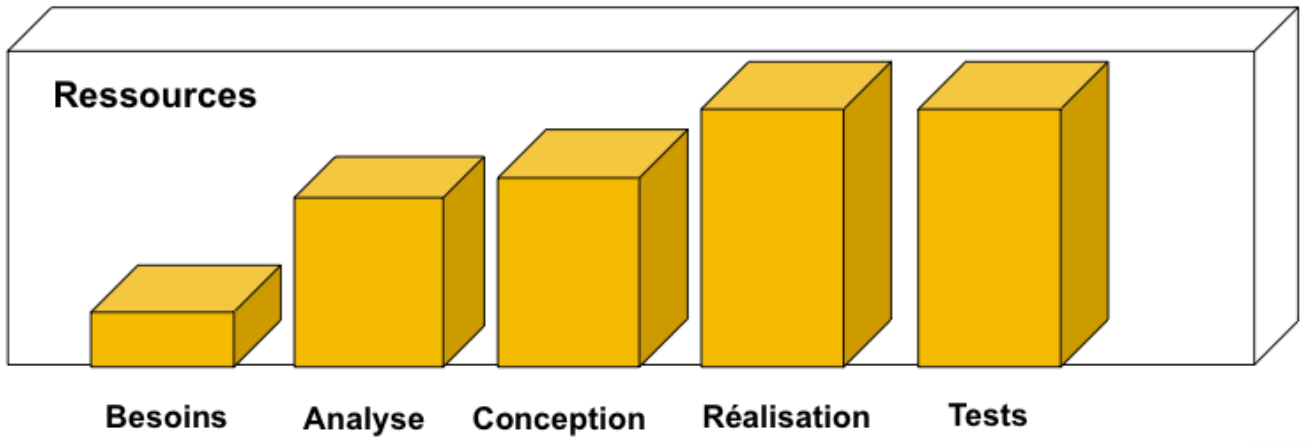


- Un modèle de l'entreprise ou du domaine complet
- Une version des modèles :
  - cas, analyse et conception (<10%)
  - Déploiement
  - implémentation (<10%)
- Une architecture de base exécutable
- La description de l'architecture (extrait des autres modèles)
  - document d'architecture logicielle
  - 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

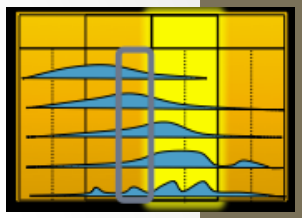
Phase de construction



- Objectif : réaliser une version beta

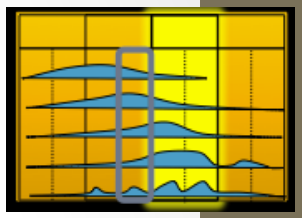


Activités principales de la construction



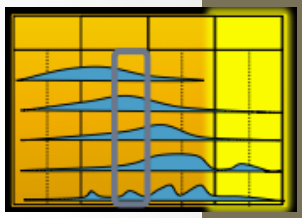
- Capture des besoins
  - spécifier l'interface utilisateur
- Analyse
  - terminer l'analyse de tous les cas d'utilisation
  - la construction du modèle structurel d'analyse
  - le découpage en paquetages…
- Conception
  - l'architecture est fixée et il faut concevoir les sous-systèmes dans l'ordre de 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

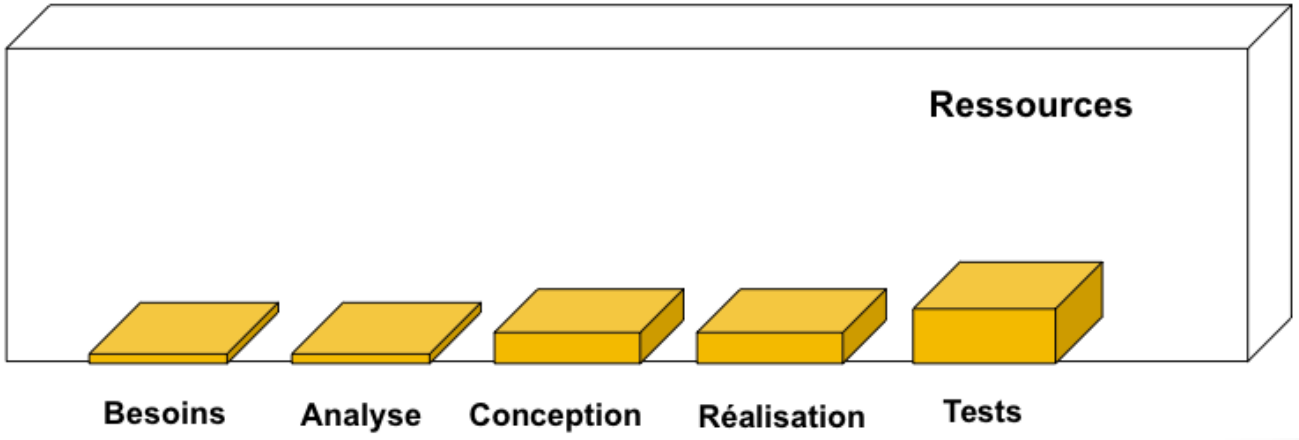


- Un plan du projet pour la phase de transition
- L'exécutable
- Tous les documents et les modèles du système
- Une description à jour de l'architecture
- Un manuel utilisateur suffisamment détaillé pour les tests

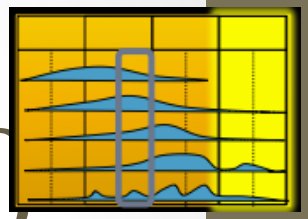
Phase de transition



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

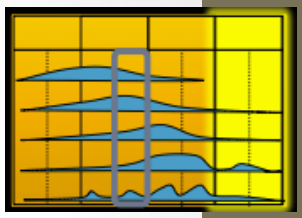


Activités principales de la transition (déploiement)



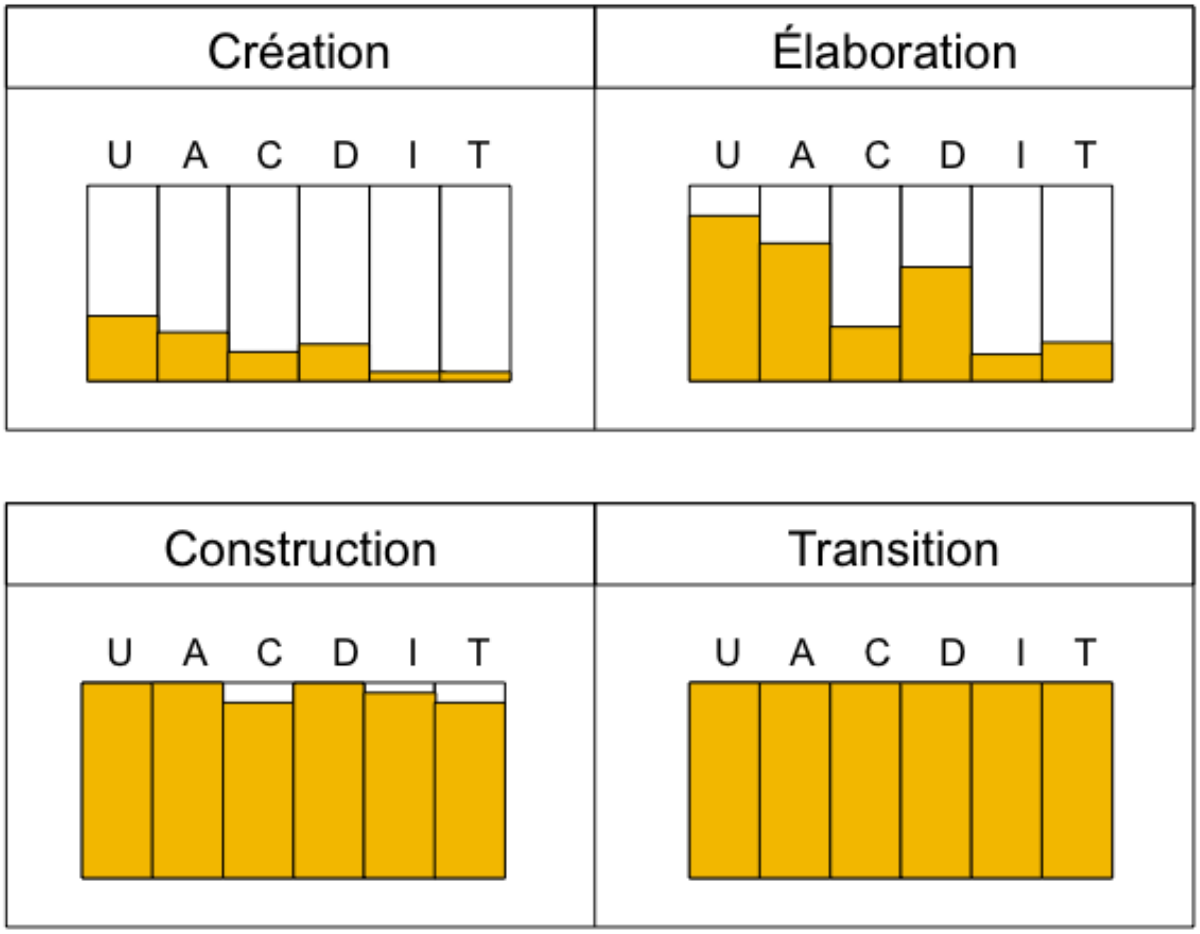
- Préparer la version bêta à tester
- Installer la version sur le site, convertir et faire migrer les données nécessaires…
- Gérer le retour des sites
- Le système fait- il ce qui était attendu ? Erreurs découvertes ?
- Adapter le produit corrigé aux contextes utilisateurs (installation...)
- Terminer 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…)

Taux de complétion des modèles par rapport aux phases



U : modèle des cas d'utilisation

A : modèle d'analyse

C : modèle de conception

D : modèle de déploiement

I : modèle d'implémentation

T : modèle de tests

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
    - Activités, travailleurs, artefacts & modèles
    - Différentes activités pour passer des besoins au code
    - Différentes phases pour piloter les activités
    - Quelques remarques
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Attention à 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

- 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

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
  - Trame de processus unifié
  - Caractéristiques essentielles
  - Description du processus unifié
  - Conclusion
- Méthodes Agile
- Conclusion
- Bonus

Conclusion sur UP

- UP décrit un ensemble de processus applicables
  - Il faut les adapter aux besoins du projet en cours

- UP a été décliné
  - TTUP : Two Tracks Unified Process
  - UP Agile
  - …

  ⇒ à vous de creuser ☺

- Les principes fondamentaux de UP sont valables pour toute conception orientée-objets
  - Y compris les méthodes dites « Agiles » ….

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
- Méthodes Agile
  - Principes
  - XP
- Conclusion
- Bonus

Historique

- 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 »
- Années 2000
  - projets Agile mixent des éléments des principales méthodes

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 1991, 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…

Manifeste Agile : principes

1. La plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à forte valeur ajoutée.
2. Le changement est accepté, même tardivement dans le développement, car les processus agiles exploitent le changement comme avantage compétitif pour le client.
3. La livraison s'applique à une application fonctionnelle, toutes les deux semaines à deux mois, avec une préférence pour la période la plus courte.
4. Le métier et les développeurs doivent collaborer régulièrement et de préférence quotidiennement au projet.
5. Le projet doit impliquer des personnes motivées. Donnez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance quant au respect des objectifs.
6. La méthode la plus efficace de transmettre l'information est une conversation en face à face.

Manifeste Agile : principes

7. L'unité de mesure de la progression du projet est un logiciel fonctionnel (ce qui exclut de comptabiliser les fonctions non formellement achevées).
8. Les processus agiles promeuvent un rythme de développement soutenable (afin d'éviter la non qualité découlant de la fatigue).
9. Les processus agiles recommandent une attention continue à l'excellence technique et à la qualité de la conception.
10. La simplicité et l'art de minimiser les tâches parasites, sont appliqués comme principes essentiels.
11. Les équipes s'auto-organisent afin de faire émerger les meilleures architectures, spécifications et conceptions.
12. À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son processus de travail en conséquence.

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

De multiples méthodes

- XP (Kent Beck)
- Scrum (Ken Schwaber)
- Crystal (Alistair Cockburn)
- FDD (Feature Driven Development)
- DSDM (Dynamic System Development Method)
- …

- Nous allons étudier rapidement XP…
  … à vous de creuser les autres ☺

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
- Méthodes Agile
  - Principes
  - XP
- Conclusion
- Bonus

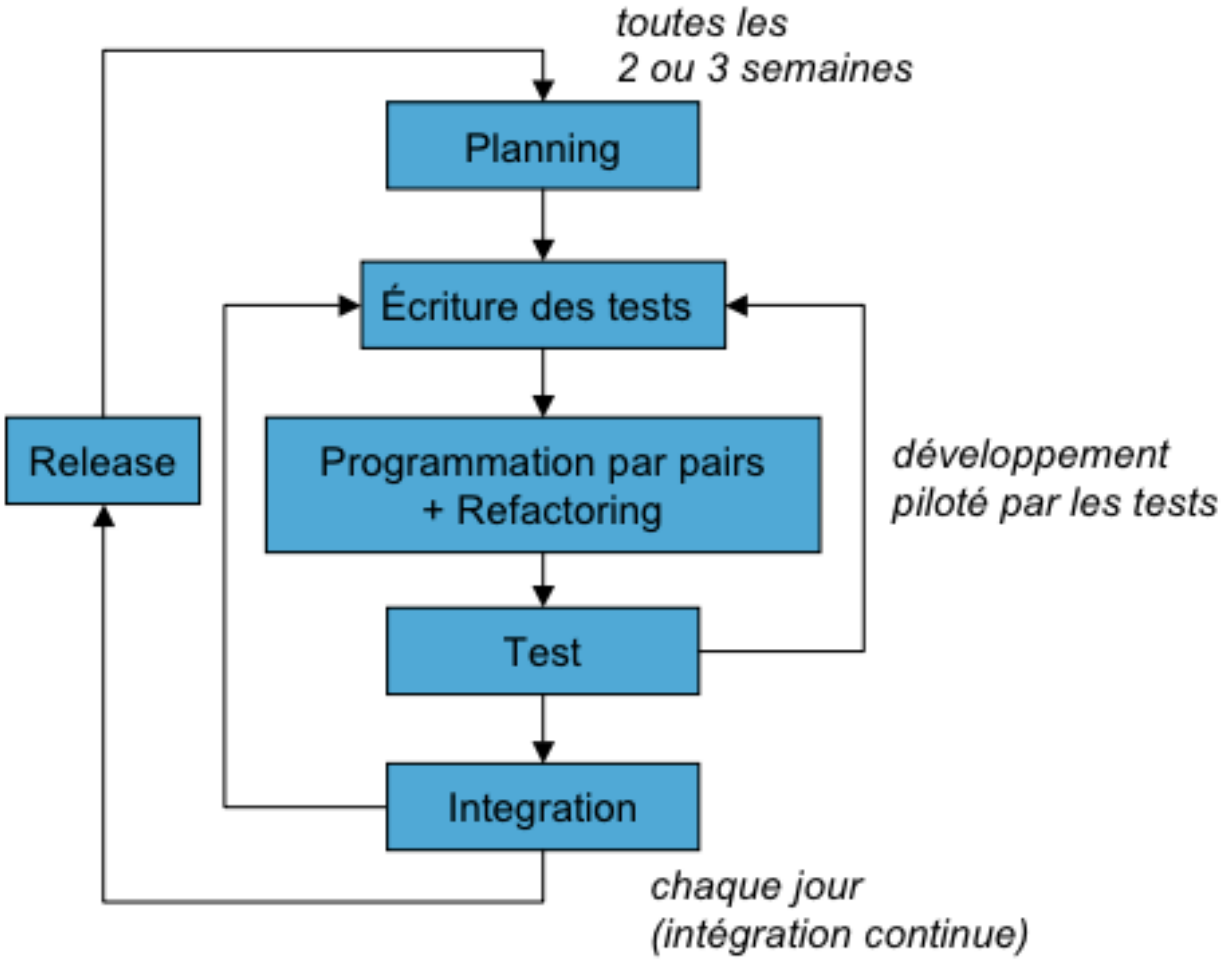
eXtreme Programming

- Historique
  - 1996 : Kent Beck, Ward Cunningham et Ron Jeffries
  - Projet Chrysler Comprehensive Compensation (C3)
  - Réorganisation du système de paie
  - Propagation mondiale grâce à Internet
  - Implémentation lente en France
- Dimension humaine = déterminante pour la réussite de tout projet
- Principes
  - Feedback permanent
  - Compréhension partagée
  - Bien-être de l'équipe
  - Processus fluide et continu

XP – Caractéristiques principales

- 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.

XP – Vue d'ensemble



TDD – Test Driven Development

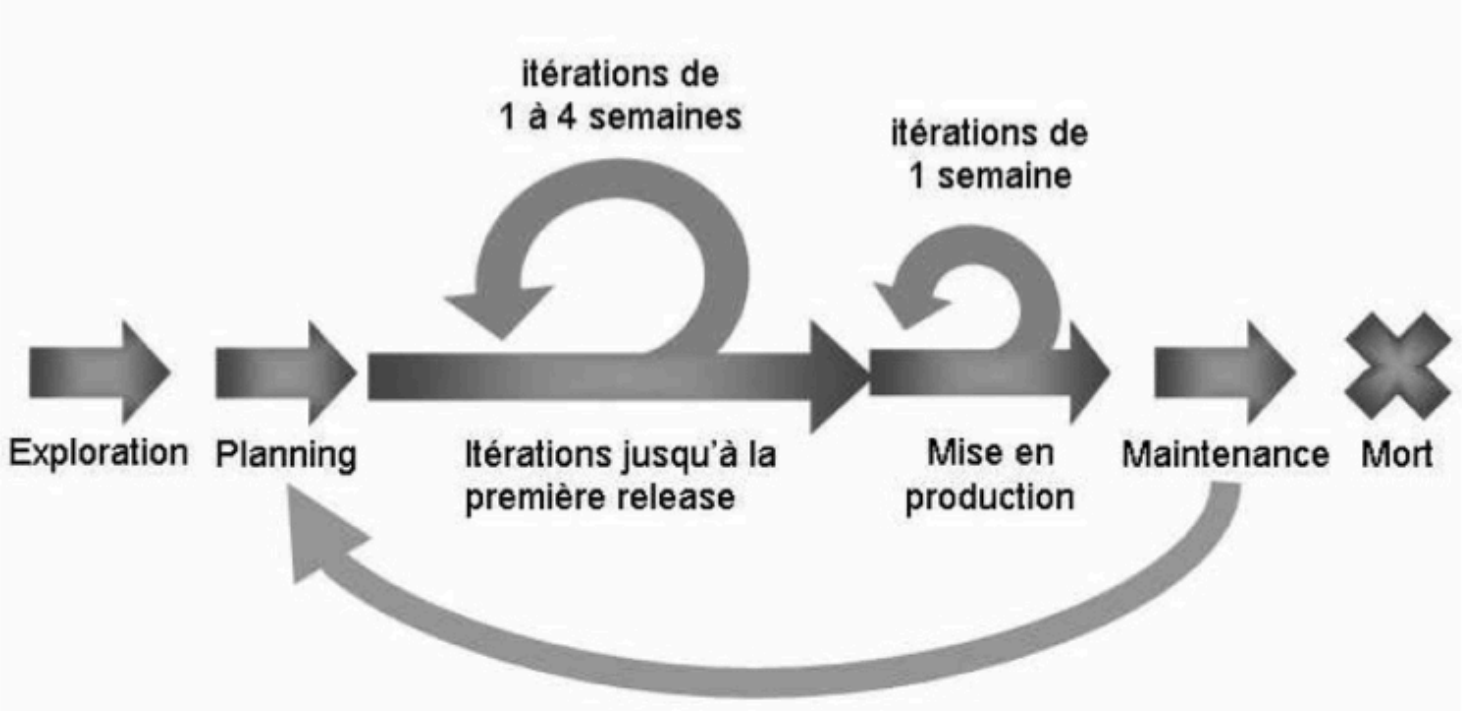
- Technique de developpement
  - Ecrire les tests unitaires avant d'ecrire le source d'un logiciel

- Cycle en 5 étapes
  1. écrire un premier test ;
  2. vérifier qu'il échoue (car le code qu'il teste n'existe pas), afin de vérifier que le test est valide ;
  3. écrire juste le code suffisant pour passer le test ;
  4. vérifier que le test passe ;
  5. puis améliorer tout en gardant les mêmes fonctionnalités.

TDD – Test Driven Development

- Intérêt :
  - Les tests permettent de préciser les spécifications du code
    - … et donc le comportement du logiciel
  - On est sur de pouvoir tester unitairement un code
    - Soulève les éventuelles erreurs de conception
  - Construction conjointe d'un programme et des tests de non-régression
    - Lors de l'amélioration du code, les tests valident doivent le rester
- Bien adapté à XP et à la programmation par binôme
  - Un développeur code les tests, l'autre la fonctionnalité

XP – Planification du projet



XP – Structure de l'équipe

- Le client
  - Membre à part entière de l'équipe
  - Présence physique imposé tout au long du développement
  - Spécifie les fonctionnalités à implémenter tout au long du développement
  - Rôle pouvant être tenu par une ou plusieurs personnes
- Le testeur
  - Assistant du client
  - Un programmeur
  - Implémente les tests de recette le plus tôt possible, valide le fonctionnement du code et vérifie la non-regression

XP – Structure de l'équipe

- Le manager
  - Responsable de l'infrastructure dans laquelle l'équipe travaille
  - S'assure le non existence de problèmes étrangers au projet ou logistiques (espace de travail, outillage, documentation, etc.)

- Le coach
  - S'assure de la bonne compréhension de la méthode XP et de son application correcte par les différents acteurs du projet
  - Généralement « expert méthode » et bon communicateur doublé d'un technicien crédible et respecté
  - A pour objectif de se faire oublier

XP – Structure de l'équipe

- Le tracker
  - Contrôle l'avancement des tâches à l'intérieur d'une itération
  - S'entretient fréquemment avec chaque programmeur pour s'enquérir des difficultés rencontrées et du travail déjà effectué
  - Construit régulièrement une vision fiable de l'avancement des tâches afin de détecter le plus tôt possible les dérives et de lancer une nouvelle phase si nécessaire

- Le programmeur
  - Estime la charge nécessaire à l'implémentation d'un scénario dans le cadre du jeu de la planification
  - Implémente les scénarios en s'appuyant sur l'écriture des tests unitaires
  - Est aussi analyste, concepteur.

XP - Pratiques

- 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

- 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

- 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
  - permet de
    - Se contrôler mutuellement
    - Diminuer les erreurs de conception
    - Mieux se concentrer sur le travail
    - Eliminer le risque de dépendance à un développeur
    - Lisser les inégalités d'expérience en associant « confirmé & débutant »
    - Faire circuler la connaissance à l'intérieur du projet

XP - Pratiques

- 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
  - tous les changements sont intégrés dans le code de base au minimum chaque jour : une construction complète (build) minimum chaque jour
  - 100% des tests doivent passer avant et après l'intégration

XP - Pratiques

- 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

- 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 - 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
- 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

Plan

- Qu'est-ce qu'un SI ?
- Concevoir un SI
- Processus unifié
- Méthodes Agile
- Conclusion
- Bonus

Conclusion

- Il n'y a pas de solution miracle
  - 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.

Bonus ☺

- La méthode R.A.C.H.E :
  - Rapid
  - Application
  - Conception and
  - Heuristic
  - Extreme-programming

- http://www.risacher.com/la-rache/index.php?z=2