COO Avancée

Licence 3 Informatique Semestre 6

Philippe Renevier-Gonin
Clémentine Némo
http://deptinfo.unice.fr/~renevier/COOA

Université Nice SOPHIA ANTIPOLIS

D'après un cours de Ph. Collet

Pourquoi ?

- Connaître (l'essentiel d') UML ne suffit pas pour réaliser de bonnes conceptions
- UML n'est qu'un langage, une notation…
- En plus, il faut :
  - Savoir penser et coder en termes d'objets (cf. POO)
  - Savoir organiser l'analyse et la conception
  - À grande échelle
  - A l'aide de guides méthodologiques adaptés au(x) problème(s)
  - Sans vision dogmatique
  - Mais en réutilisant des bons principes communs

Objectif de l'UE

- Vous sensibiliser à l'ingénierie des systèmes d'information.
- Vous donner une vision complète de l'activité de conception au sens large (analyse, conception, spécification) dans le cycle de développement logiciel.
- Liaison IHM (maquette) – Système Information
- Vous faire mettre en pratique les différentes activités qui constituent l'étape de conception d'un processus de développement.
- Vous faire travailler en équipe, mais aussi à plusieurs équipes communicantes !

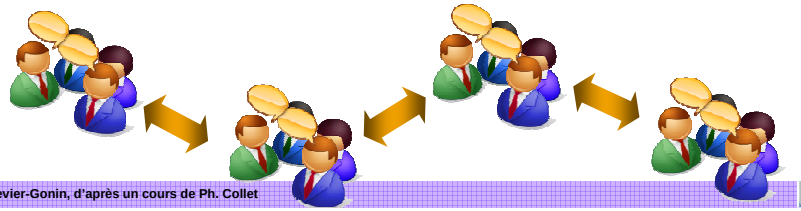
Programme

- Introduction et motivations
- Rappels sur UML et les activités d'analyse et de conception
- Processus de conception : définition
- Processus Unifié UML : RUP
- Processus agiles
- Conclusion

- 3 cours : mercredi 8h-10h
- 1er TD : 8h30 vendredi 28 en M I - 3

Organisation

- **Crédit**
  - 4 ECTS ou Coeff 4
- **Enseignement**
  - 6h de cours
  - 30h de TD
- **Evaluation**
  - Projet tout le long de l'enseignement par équipe de 4 ou 5
  - 2 rendus : à mi parcours (vacances de février) et final – 40% chacun
  - Soutenance – 20 %



Rappels

(ou pas)

Génie logiciel

- Définition (*software engineering*)
  - ensemble de méthodes, techniques et outils pour la production et la maintenance de composants logiciels de qualité
- Principes
  - rigueur et formalisation, séparation des préoccupations, modularité, abstraction, prévision du changement, approche générique, approche incrémentale
- Besoins
  - Langages *pour décrire*
  - Outils *pour manipuler*
  - **Méthodes** *pour décider*
  - Théories *pour démontrer*
  - Professionnels *pour réaliser*
  - Logistique *pour supporter*

Qualités pour l'utilisateur (phases d'exploitation)

- **Fiabilité = Validité + Robustesse**
  - Validité ≡ correction, exactitude : assurer exactement les fonctions attendues, définies dans le cahier des charges et la spécification, en supposant son environnement fiable
  - Robustesse: faire tout ce qu'il est utile et possible de faire en cas de défaillance: pannes matérielles, erreurs humaines ou logicielles, malveillances…
- **Performance**
  - Utiliser de manière optimale les ressources matérielles : temps d'utilisation des processeurs, place en mémoire, précision…
- **Convivialité**
  - Réaliser tout ce qui est utile à l'utilisateur, de manière simple, ergonomique
- **Extensibilité, Compatibilité, Intégrité (sécurité)…**

Qualités pour le développeur (phases de devt)

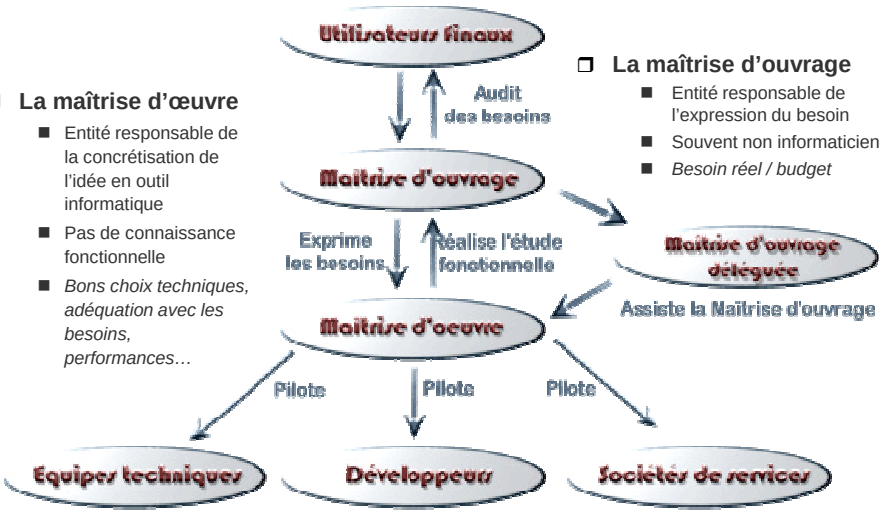
- **Documentation**
  - Tout ce qu'il faut, rien que ce qu'il faut, là où il faut, quand il faut, correcte et adaptée au lecteur
- **Modularité = Fonctionnalité + Interchangeabilité + Réutilisabilité**
  - Fonctionnalité : Localiser un phénomène unique, facile à comprendre et à spécifier
  - Interchangeabilité : Pouvoir substituer une variante d'implémentation sans conséquence fonctionnelle (et souvent non-fonctionnelle) sur les autres parties
  - Réutilisabilité : Aptitude à être réutilisé, en tout ou en partie, tel que ou par adaptation, dans un autre contexte : autre application, machine, système…
- **Vérifiabilité**
  - Aptitude d'un logiciel à être testé
- **Portabilité**
  - Aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents

Génie logiciel : le défi

- **Contradictions apparentes**
  - Qualités vs coût du logiciel
  - Qualités pour l'utilisateur vs qualités pour le développeur
  - Contrôler vs produire
- **Conséquences**
  - Chercher sans cesse le meilleur compromis
  - Amortir les coûts
    - Premier exemplaire de composant coûteux à produire ou à acheter, puis amortissement…
  - Gérer un projet informatique de manière spécifique (avec des **méthodes** spécifiques)

Organisation d'un projet informatique

- **La maîtrise d'œuvre**
  - Entité responsable de la concrétisation de l'idée en outil informatique
  - Pas de connaissance fonctionnelle
  - *Bons choix techniques, adéquation avec les besoins, performances…*

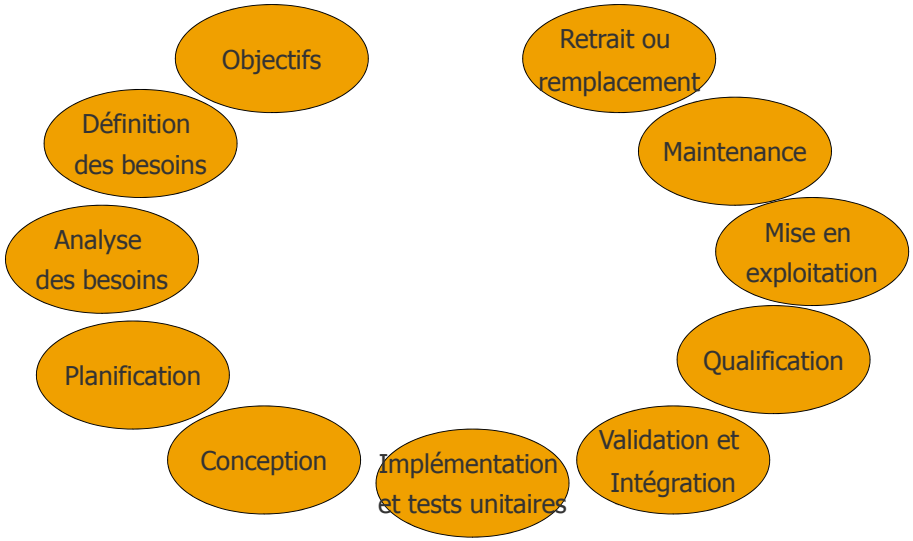


- **La maîtrise d'ouvrage**
  - Entité responsable de l'expression du besoin
  - Souvent non informaticien
  - *Besoin réel / budget*

Comment gérer un projet informatique ?

- **Définir et utiliser des méthodes**
  - spécifiant des processus de développement
  - organisant les activités du projet
  - définissant les artefacts du projet
  - se basant sur des modèles
    - pas des modèles objets, des modèles de **cycle de vie**
- **Modèles de cycle de vie**
  - organiser les différentes phases du cycle de vie pour l'obtention d'un logiciel fiable, adaptable et efficace, etc.
  - guider le développeur dans ses activités techniques
  - fournir des moyens pour gérer le développement et la maintenance

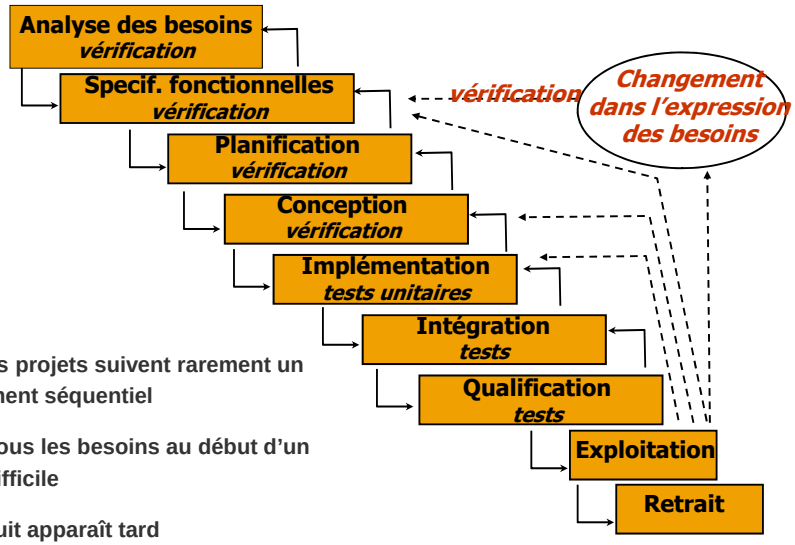
Les phases du cycle de vie



Mais c'est quoi un « cycle de vie » ?

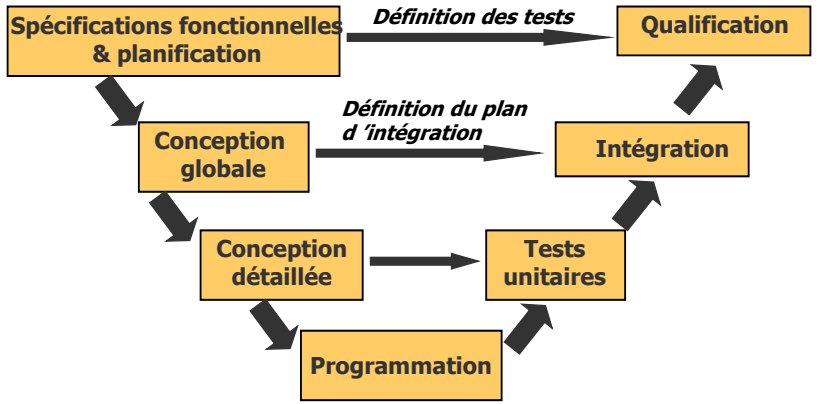
- **Cycle de vie : définition**
  - Description d'un processus pour la création d'un produit, sa distribution sur un marché, son retrait
- **Cycle de vie et assurance qualité**
  - Validation : le bon produit ?
  - Vérification : le produit correct ?
  - Le cycle de vie s'organise autour de l'assurance qualité
- **Deux types principaux de modèles**
  - Modèle linéaires : en cascade et variantes
  - Modèles non linéaires : en spirale, incrémentaux, itératifs

Modèle en cascade (1970)



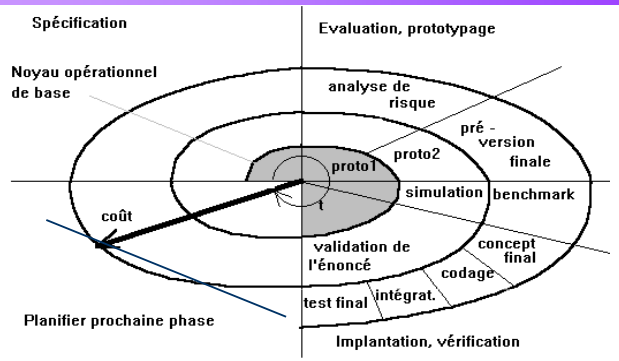
- Les vrais projets suivent rarement un développement séquentiel
- Établir tous les besoins au début d'un projet est difficile
- Le produit apparaît tard
- Echecs majeurs sur de grands systèmes

Modèle en V



- Meilleure anticipation
- Bonne structuration des tests
- Cadre de développement rigide
- Le produit apparaît toujours tard

Modèle en spirale (Boehm, 1988)



- **Incréments successifs => itérations**
  - Approche souvent à base de prototypes
  - Nécessite de bien spécifier les incréments
  - Figement progressif de l'application
- **Mais gestion de projet pas évidente**
- **Pourtant, les méthodes objets dérivent de ce modèle !**

Méthodes et processus de conception

Université Nice SOPHIA ANTIPOLIS

Définitions

- **Modèle (comme UML)**
  - Notation pour représenter, formalisme
  - Pour partager l'information, uniformiser, mécaniser (transformation vers le code)
- **Méthode**
  - Guide plus ou moins formalisé
  - Démarche reproductible permettant d'assurer la qualité
- **Processus**
  - A peu près la même définition qu'une méthode…
  - « ensemble de directives et 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 »

Définitions (suite)

- **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 (quand construit-on les modèles)
  - un ensemble de règles et de conseils
- **Une méthode peut donc reposer sur un processus…**
  - **= notation + artefact (ce qui est manipulé) + démarche**
- **On distingue souvent :**
  - **Les processus de développement technique**
  - **Les processus de gestion du développement lui-même**

Evolution des méthodes

- **Premières méthodes** (années 60-70)
  - Guide par découpage en sous-problèmes, analyse fonctionnelle
  - Méthodes d'analyse structurée
- **Méthodes orientées bases de données** (années 80 et suivantes)
  - Concevoir la base de données est central au système d'information
  - Méthodes globales qui séparent données et traitements
- **Méthodes pour la conception avec réutilisation** (années 95 et suivantes)
  - Frameworks, Design Patterns, bibliothèques de classes (orientées objets)
  - Méthodes incrémentales unifiées par une notation commune (UML)

Quelles activités sont couvertes par les méthodes ?

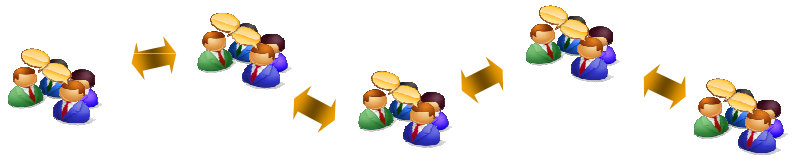
- Cinq grandes activités émergent de la pratique de développement et de gestion de projet
  - Spécification des besoins
  - Analyse (recherche du bon système)
  - Conception (liée à l'implémentation, la construction)
  - Implémentation (activité la plus coûteuse)
  - Tests (activité la plus négligée)

Organisation des TD

Université Nice SOPHIA ANTIPOLIS

TD, équipes, sujets…

- **Jusqu'à 30 personnes par TD**
- **6 équipes de 4 pour 5 sujets**
- **Les équipes sont formées par consensus par les étudiants d'un même groupe de TD**
- **Les sujets sont tirés au sort**



- **Chaque sujet a au moins un lien avec un autre sujet, il nécessitera donc une collaboration entre les deux équipes concernées, au moins au niveau des cas d'utilisation**
- **Chaque séance de TD est l'occasion d'avancer dans une partie de la modélisation**

Déroulement et évaluation

- **5 séances de TD (3h) pour démarrer la conception**
- **5 séances de TD (3h) pour terminer la conception et utiliser un outil UML afin de finaliser le dossier à rendre**
- **Portable(s)**
- **Travail itératif**
  - **RENDU A MI PARCOURS (version minimale avec au moins un échange avec un autre projet)**
  - **DOSSIER ET SOUTENANCE FINALE**
- **Le chargé de TD guide les équipes et joue le rôle de client**
- **15 avril :** Dossier rendu
- 22 avril : 1 soutenance par équipe (10/15 minutes + 10 minutes)

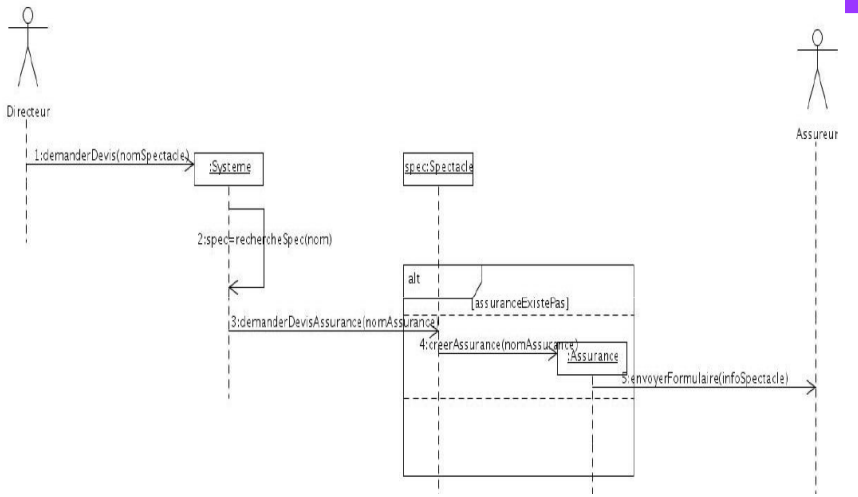
Contenu du dossier de conception

- Ensemble des diagrammes de cas d'utilisation, cohérents vis-à-vis des autres systèmes d'informations.
- Diagramme détaillé de classes (signature typée des attributs et des méthodes).
- Un diagramme de séquences par cas d'utilisation
- Une maquette de(s) IHM(s) en lien avec les diagrammes
- Au moins un diagramme d'activités (il est bien sûr pertinent de l'utiliser pour détailler un cas d'utilisation particulièrement complexe...)
- Au moins un diagramme d'états (il est bien sûr pertinent de l'utiliser pour détailler une classe dont les états sont remarquables...)
- Des spécifications OCL (préconditions, postconditions, invariants) sur les classes qui vous semblent les plus pertinentes.

Maquette de l'IHM : concrétiser la conception

- **Problème d'abstraction au niveau de la conception**
- **Faire la maquette**
  - **« Concrétiser » l'utilisation du système**
- **Y associer les use-cases**
  - **si le use case fait intervenir un acteur humain et qu'il n'y a pas d'Ihm, alors ce n'est pas un use-case**
- **Y associer diagramme de séquence**
  - **Déterminer comment les infos sont présentées, comment les paramètres sont entrés**
- **(voire Y associer diagramme de classe pour la création ou la manipulation d'objet)**
- **Vérifier la Cohérence**

Exemple d'incohérence (1/2)



Description: Le directeur envoi une demande de devis, pour un spectacle et une assurance donné, via l'interface du Système, celui ci entame alors une recherche du spectacle concerné pour ensuite envoyer une requête de demande de devis. Une instance d'assurance est alors crée pour envoyer le formulaire demandé à l'assurance concernée.

Exemple d'incohérence (2/2)

Nom du devis : Mariage PRG

Demander le devis

Système

```
+parcourirDemandeMairie()
+getListeContratAssurance()
+getListePersonnel()
+getListeEquipement()
+getListePlacesDispo()
+verifierPlacesDispo()
+demandeAutorisation()
+creerRepresentation()
+creerBillet()
+recuClient()
+envoyerErreur()
+rechercherSpec()
+ajouterSinistre()
+demandeAssurance()
+chercherDevis()
+validationDevis()
+MiseAJourPlacesDispo()
+calculDépenseReprésentation(): float
+calculBénéficeReprésentation(): float
+calculBénéficesAnnuel(): float
```

- demanderDevis( ) ?
- D'où vient le nom de l'assurance (création) ?
- Il en faut +

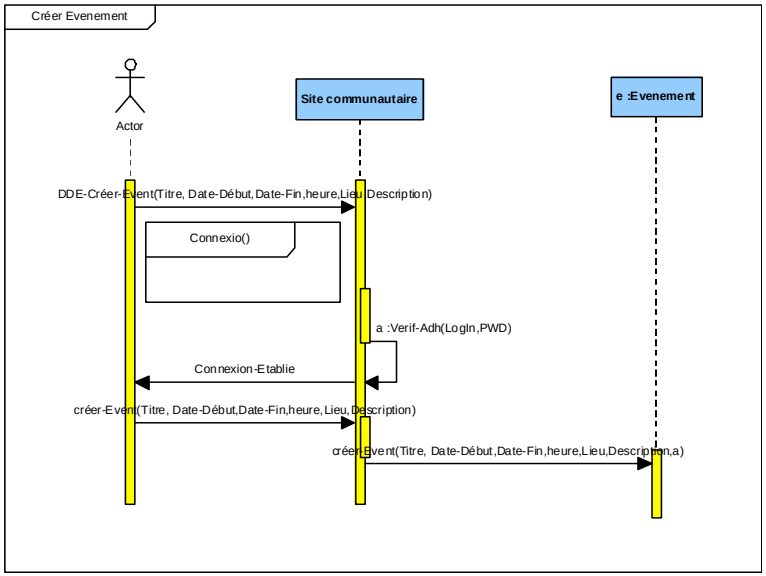
Nom du devis : Mariage PRG

Il n'y a pas d'assurance assoicée à l'événement.

Assureur : Azur Assurance ▼

Choisir Assureur

Autre exemple (1/2)



Autre exemple (2/2)

Titre :

Début : Fin :

Lieu :

Description :

Créer l'événement

DDE-Créer-Event

Login :

Password :

connexion

Retour de login

Créer-Event

Processus unifiés

- Principes
- Description
- Déclinaison

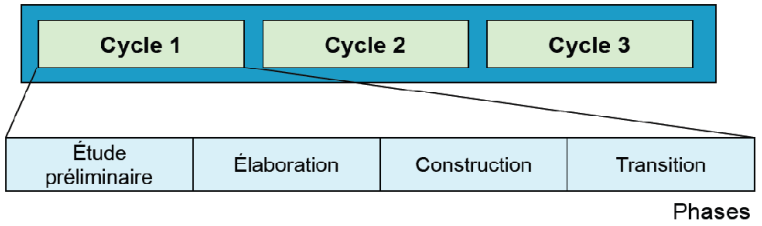
Université Nice SOPHIA ANTIPOLIS

Unified Software Development Process

- **USDP**
  - **Résumé en UP : *Unified Process***
- **Avant UML**
  - **Autant de notations que de méthodes**
  - **Focalisation sur certains aspects uniquement**
- **Avec UML**
  - **Uniformisation**
- **USDP**
  - **Processus général et méthode de conception**
  - **Pour gérer un projet de bout en bout**
  - **À décliner en fonction des notations (UML) et des processus plus particuliers utilisés**

USDP : Principes

- 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
  - chaque phase se termine par un point de contrôle (ou jalon) permettant de prendre des décisions



Phases

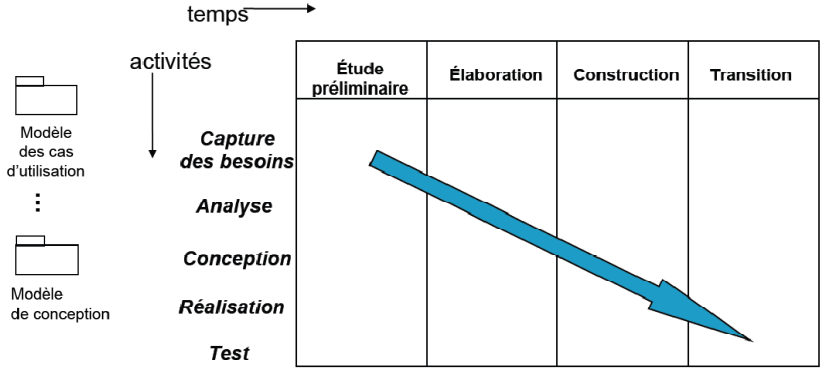
- **Phase 1 : étude préliminaire**
  - que fait le système ?
  - à quoi pourrait ressembler l'architecture ?
  - quels sont les risques ?
  - quel est le coût estimé du projet ? Comment le planifier ?
  - accepter le projet ?
  - jalon : « vision du projet »
- **Phase 2 : Élaboration**
  - spécification de la plupart des cas d'utilisation
  - conception de l'architecture de base (squelette du système)
  - mise en oeuvre de cette architecture (UC critiques, <10 % des besoins)
  - planification complète
  - besoins, architecture, planning stables ? Risques contrôlés ?
  - jalon : « architecture du cycle de vie »

Phases (suite)

- **Phase 3 : Construction**
  - développement par incréments
    - architecture stable malgré des changements mineurs
  - le produit contient tout ce qui avait été planifié
    - il reste quelques erreurs
  - produit suffisamment correct pour être installé chez un client ?
  - jalon : « capacité opérationnelle initiale »
- **Phase 4 : Transition**
  - produit livré (version bêta)
  - correction du reliquat d'erreurs
  - essai et amélioration du produit, formation des utilisateurs, installation de l'assistance en ligne
  - tests suffisants ? Produit satisfaisant ? Manuels prêts ?
  - jalon : « livraison du produit »

Phases et activités

- Le cycle met en jeu des activités
  - vue du développement sous l'angle technique
  - Les activités sont réalisées au cours des phases, avec des importances variables
  - Les activités consistent notamment à créer des modèles



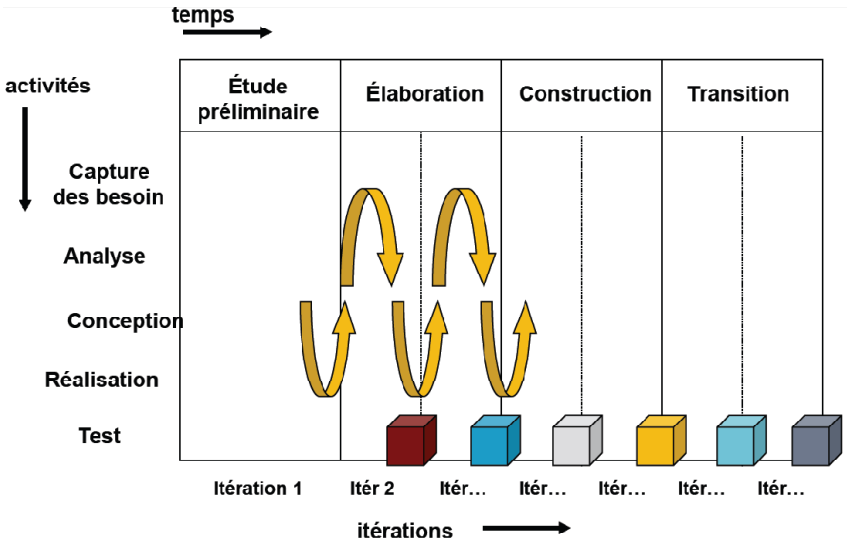
USDP : principes d'organisation

- **USDP : tout le contraire du modèle en cascade :**
  - Point commun à toutes les méthodes OO ?
    - le changement est… **constant**
  - Feedback et adaptation : décisions nécessaires tout au long du processus
  - Convergence vers un système satisfaisant
- **Principes résultants :**
  - construction du système par incréments
  - gestion des risques
  - passage d'une culture produit à une culture projet
  - « souplesse » de la démarche

Itérations et incréments

- **Des itérations**
  - chaque phase comprend des itérations
  - une itération a pour but de maîtriser une partie des risques et apporte une preuve tangible de faisabilité
    - produit un système partiel opérationnel (exécutable, testé et intégré) avec une qualité égale à celle d'un produit fini
    - qui peut être évalué (va-t-on dans la bonne direction ?)
- **Un incrément par itération**
  - le logiciel et le modèle évoluent suivant des incréments
  - série de prototypes qui vont en s'améliorant
    - de plus en plus de parties fournies
    - retours utilisateurs
  - processus incrémental
  - les versions livrées correspondent à des étapes de la chaîne des prototypes

Itérations dans les phases



Gestion du risques dans les itérations

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

Avantages (et inconvénients) résultants

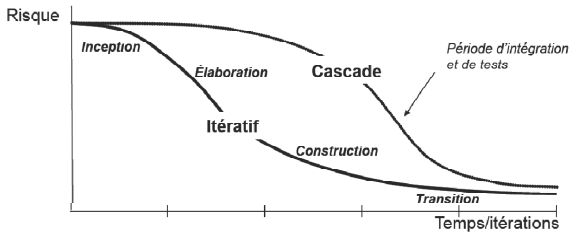
- Gestion de la complexité
- Maîtrise des risques élevés précoce
- Intégration continue
- Prise en compte des modifications de besoins
- Apprentissage rapide de la méthode
- Adaptation de la méthode
- **Mais gestion de projet plus complexe : planification adaptative**

Pilotage du processus (par les cas d'utilisation)

- Objectif du processus
  - construction d'un système qui réponde à des besoins
  - par construction complexe de modèles
- Les cas d'utilisation spécifient le besoin à travers les objectifs utilisateurs
- Ils sont utilisés tout au long du cycle (ce qu'on fait naturellement en UML)
  - validation des besoins / utilisateurs
  - point de départ pour l'analyse (découverte des objets, de leurs relations, de leur comportement) et la conception
  - guide pour la construction des interfaces
  - guide pour la mise au point des plans de tests
- Les UC assurent la traçabilité !

Pilotage du processus (par la gestion des risques)

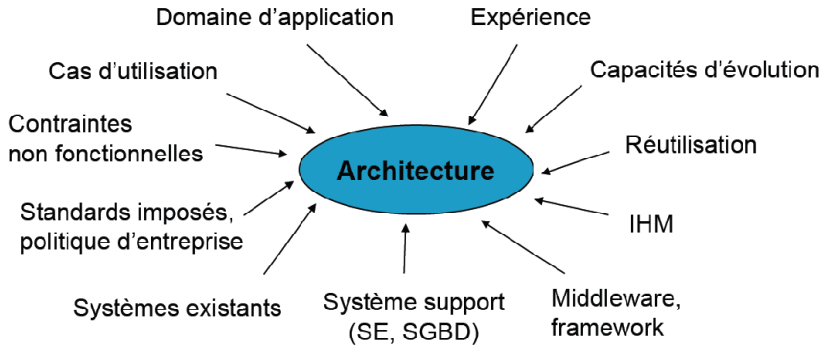
- Différentes natures de risques
  - Adéquation aux besoins, infrastructure technique, performances, personnels impliqués…
- Gestion des risques
  - identifier et classer les risques par importance
  - agir pour diminuer les risques
  - s'ils sont inévitables, les évaluer rapidement (au plus tôt)
- Construire les itérations en fonction des risques
  - provoquer des changements précoces pour stabiliser l'architecture rapidement



Processus centré sur l'architecture

- **Architecture**
  - Principe de génie civil difficile à adapter à l'informatique
- **Définition dans notre cadre**
  - Art d'assembler des composants en respectant des contraintes, ensemble des décisions significatives sur
    - l'organisation du système
    - les éléments qui structurent le système
    - la composition des sous-systèmes en systèmes
    - le style architectural guidant l'organisation (couches…)
  - Ensemble des éléments de modélisation les plus signifiants qui constituent les fondations du système à développer

Influences sur l'architecture



Différents aspects d'une architecture

- **Architecture logicielle**
  - organisation à grande échelle des classes logicielles en packages, sous-systèmes et couches
    - architectures client/serveurs en niveaux (tiers)
  - architectures en couches
  - architecture à base de composants
  - patrons architecturaux (cf. cours de Master)
- **Architecture de déploiement**
  - décision de déploiement des différents éléments
  - déploiement des fonctions sur les postes de travail des utilisateurs

Processus centré sur l'architecture

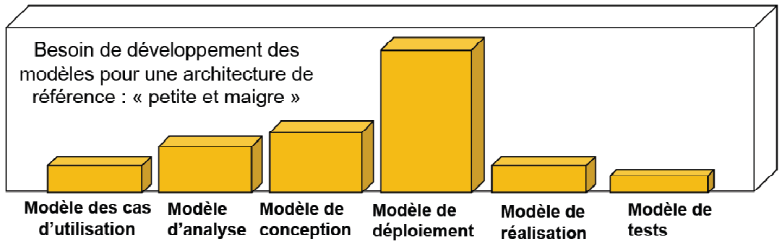
- L'architecture sert de lien pour l'ensemble des membres du projet
  - réalisation concrète de prototypes incrémentaux qui « démontrent » les décisions prises
  - vient compléter les cas d'utilisation comme « socle commun »
- Contrainte de l'architecture
  - plus le projet avance, plus l'architecture 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

Construction de l'architecture : principes

- **Démarrage, choix d'une architecture de haut-niveau et construction des parties générales de l'application**
  - ébauche à partir
    - de solutions existantes
    - de la compréhension du domaine
    - de parties générales aux applications du domaine (quasi-indépendant des CU)
  - des choix de déploiement
- **Ensuite construction de l'architecture de référence**
  - confrontation à l'ébauche des cas d'utilisation les plus significatifs (un à un)
  - construction de parties de l'application réelle (sous-systèmes spécifiques)
  - stabilisation de l'architecture autour des fonctions essentielles (sous-ensemble des CU)
  - traitement des besoins non fonctionnel dans le contexte des besoins fonctionnels
  - identification des points de variation et les points d'évolution les plus probables
- **Finalement réalisation incrémentale des cas d'utilisation**
  - itérations successives
  - l'architecture continue à se stabiliser sans changement majeur

Elaboration de l'architecture

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



Description de l'architecture

- L'architecture doit être une vision partagée
  - sur un système très complexe
  - pour guider le développement
  - tout en restant compréhensible
- **Description architecture = restriction du modèle**
  - extraits les plus significatifs des modèles de l'architecture de référence
  - vue architecturale du modèle des CU : quelques CU
  - vue du modèle d'analyse (éventuellement non maintenue)
  - vue du modèle de conception : principaux sous-systèmes et interfaces, collaborations
  - vue du modèle de déploiement : diagramme de déploiement
  - vue du modèle d'implémentation : artefacts