Processus unifiés

- Principes
- Description
- Déclinaison

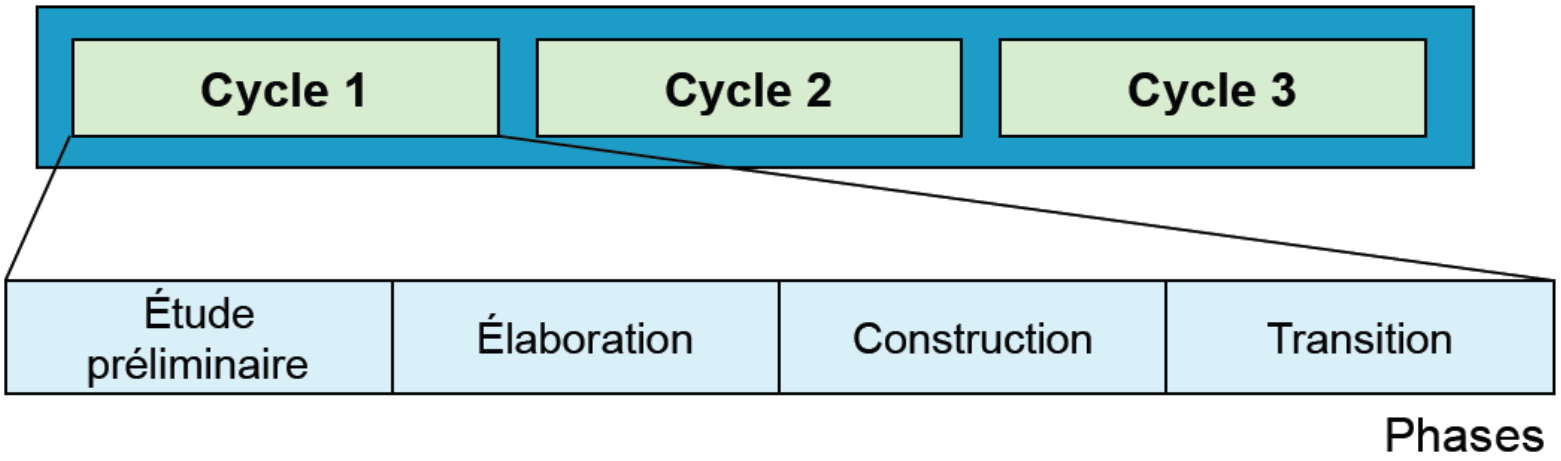
Université Nice Sophia Antipolis

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

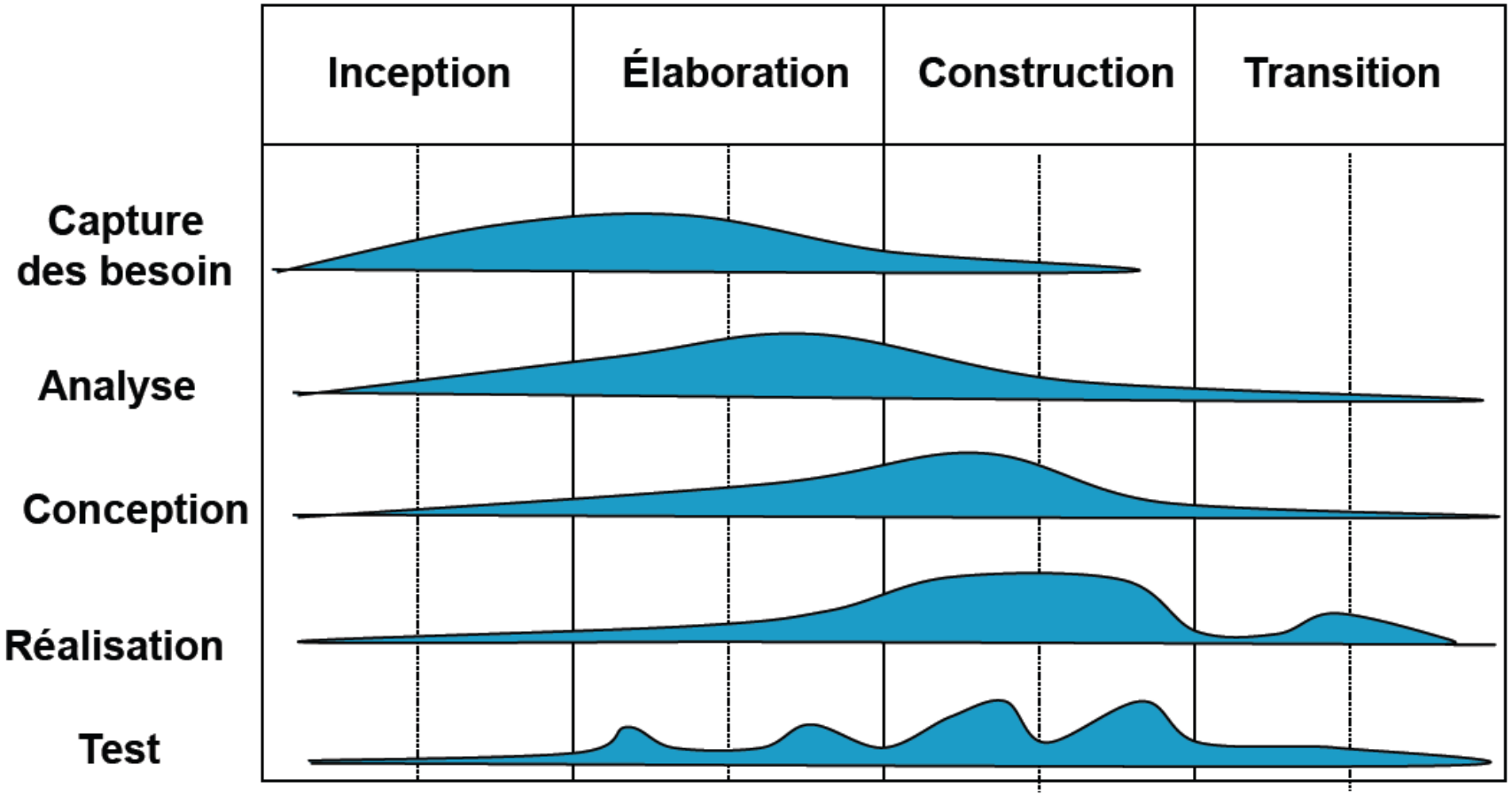
USDP : Principes

- 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



Rappel sur les phases

- Chaque phase spécifie les activités à effectuer

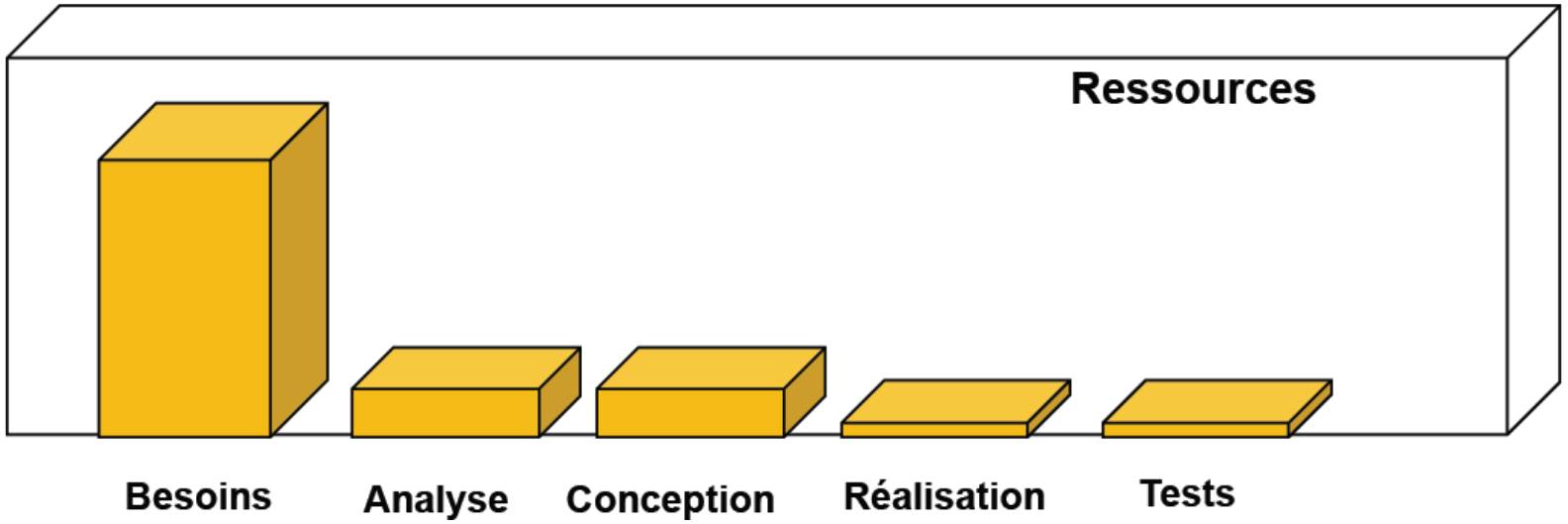


Gestion des phases

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

Phase d'étude préliminaire (inception)

- **Objectif : lancer le projet**
  - établir les contours du système et spécifier sa portée
  - définir les critères de succès, estimer les risques, les ressources nécessaires et définir un plan (petite planification)
  - **à la fin de cette phase, on décide de continuer ou non**
  - attention à ne pas définir tous les besoins, à vouloir des estimations fiables (coûts, durée), sinon on fait de la *cascade*



Activités (principales) de cette phase

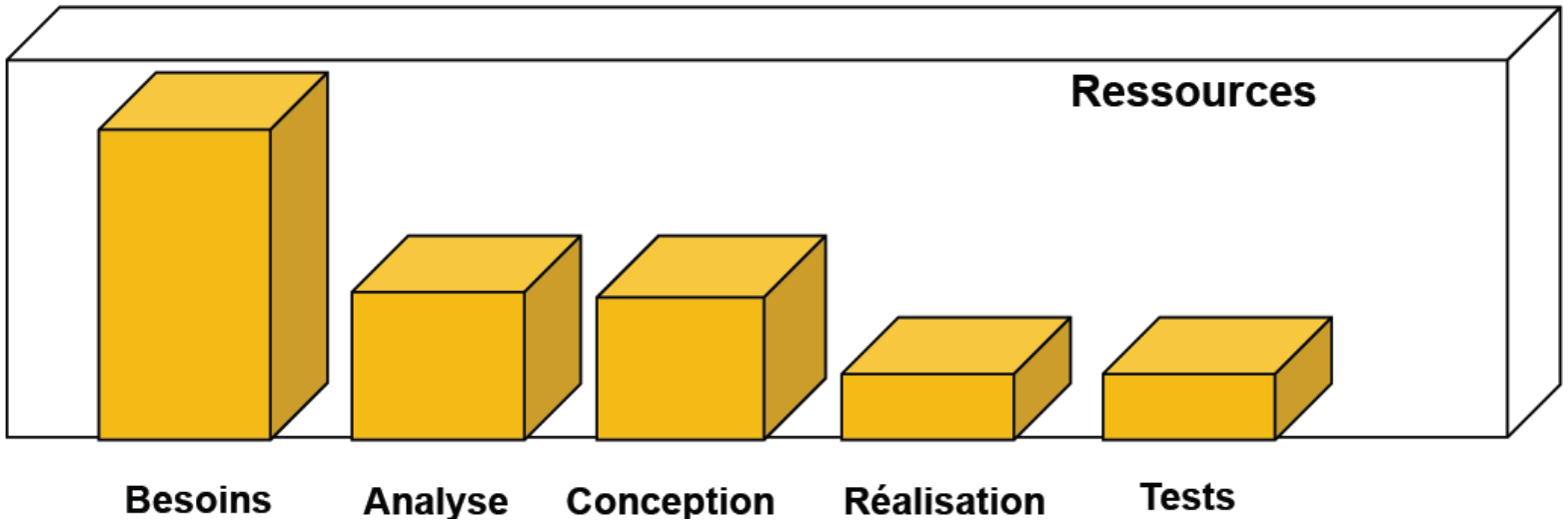
- **Capture des besoins**
  - comprendre le contexte du système (50 à 70% du contexte)
  - établir les besoins fonctionnels et non fonctionnels (80%)
  - traduire les besoins fonctionnels en cas d'utilisation (50%)
  - détailler les premiers cas par ordre de priorité (10% max)
- **Analyse**
  - analyse 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 : soussystèmes, noeuds, réseau, couches logicielles
  - examen des aspects importants et à plus haut risque

Livrables de cette phase

- 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

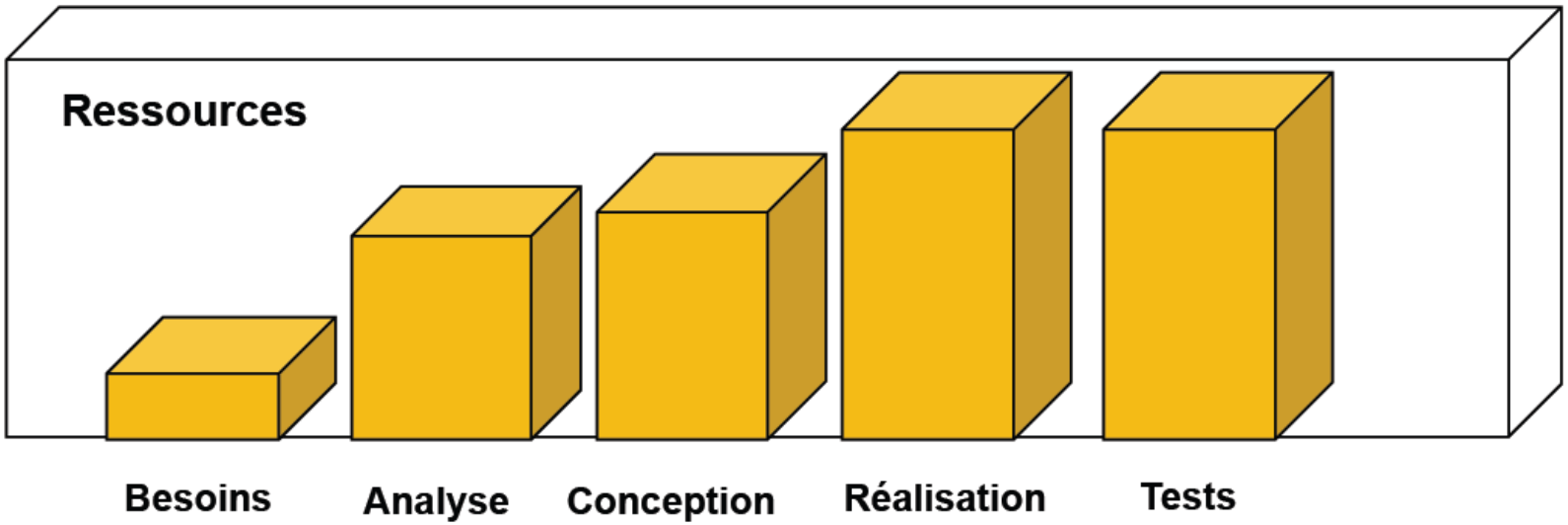
- **Capture des besoins**
  - terminer la capture des besoins et en détailler de 40 à 80%
  - faire un prototype de l'interface utilisateur (éventuellement)
- **Analyse**
  - analyse architecturale complète (packages...)
  - raffinement des cas d'utilisation (pour l'architecture, < 10%)
- **Conception**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

Livrables de l'élaboration

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

Phase de construction

- Objectif : Réaliser une version beta

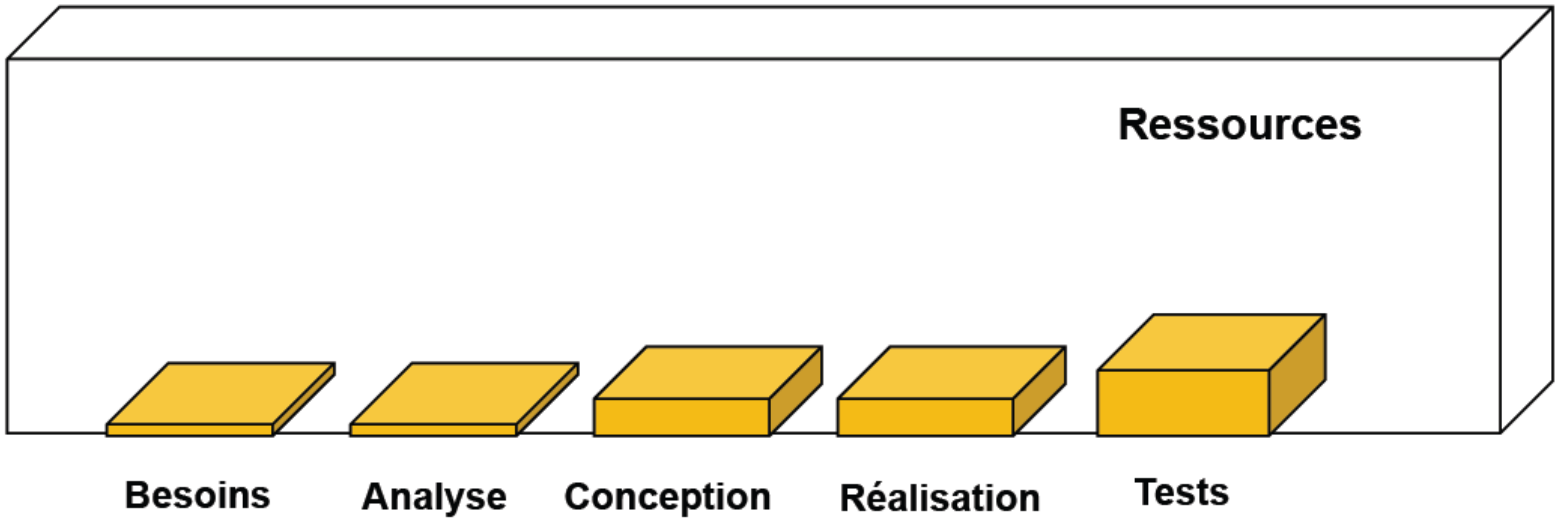


Activités et livrables de la construction

- **Capture des besoins**
  - spécifier l'interface utilisateur
- **Analyse**
  - terminer l'analyse de tous les cas d'utilisation, la construction du modèle structurel d'analyse
- **Conception**
  - l'architecture est fixée et il faut concevoir les sous-systèmes
  - dans l'ordre de priorité (itérations de 1 à 3 mois, max. 9 mois)
  - concevoir les cas d'utilisation puis les classes
- **Réalisation**
  - réaliser, passer des tests unitaires, intégrer les incréments
- **Test**
  - toutes les activités de test : plan, conception, évaluation...
- **Livrables**
  - Un plan du projet pour la phase de transition
  - L'exécutable et son packaging minimal
  - 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 version beta, correction des erreurs
  - préparation de la formation, la commercialisation



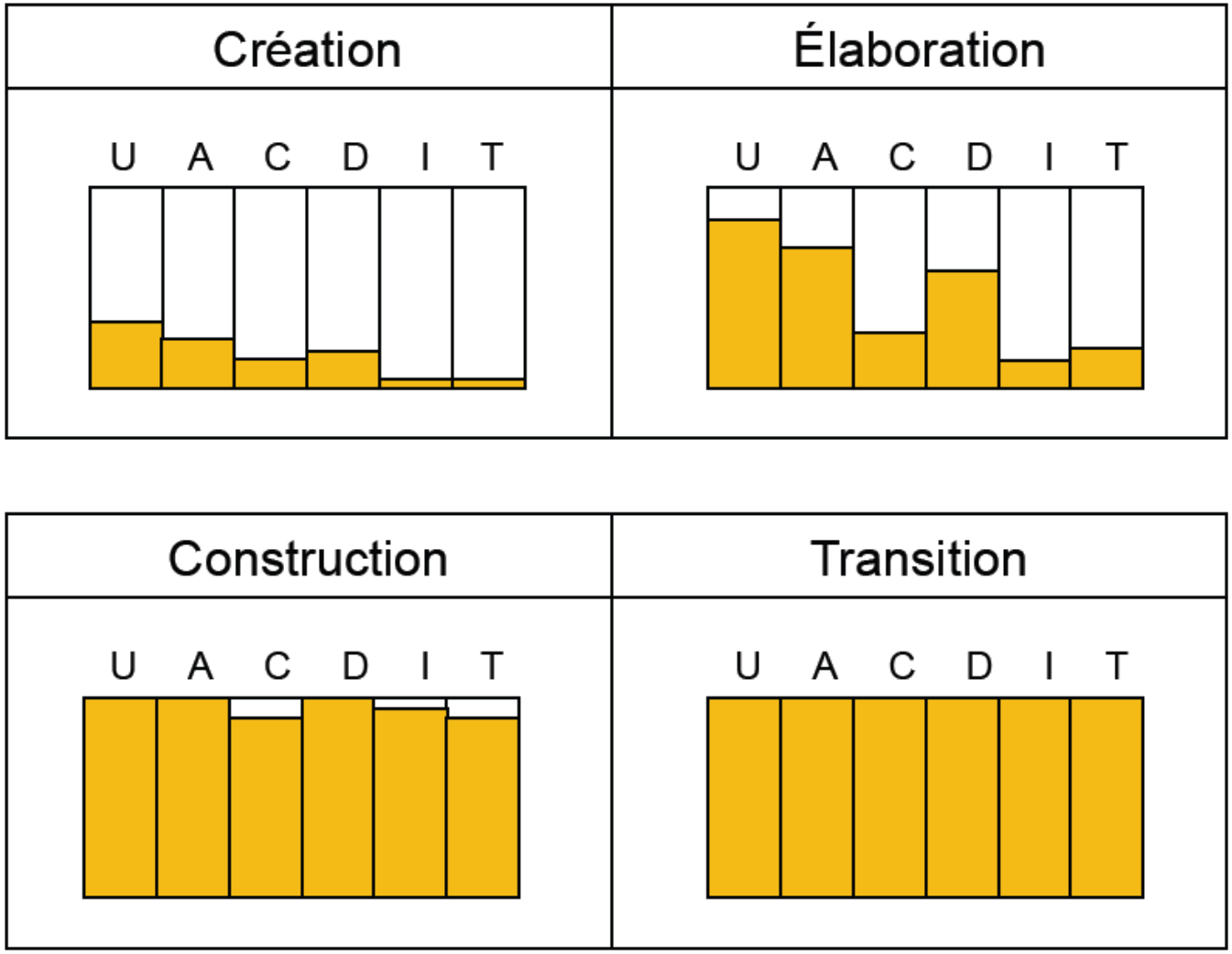
Activités principales de transition

- **Préparer la version beta à tester**
  - Installer la version sur le site, convertir et faire migrer les données
- **Gérer le retour des sites (retour de déploiement)**
  - 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...)

Répartition modèles/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

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

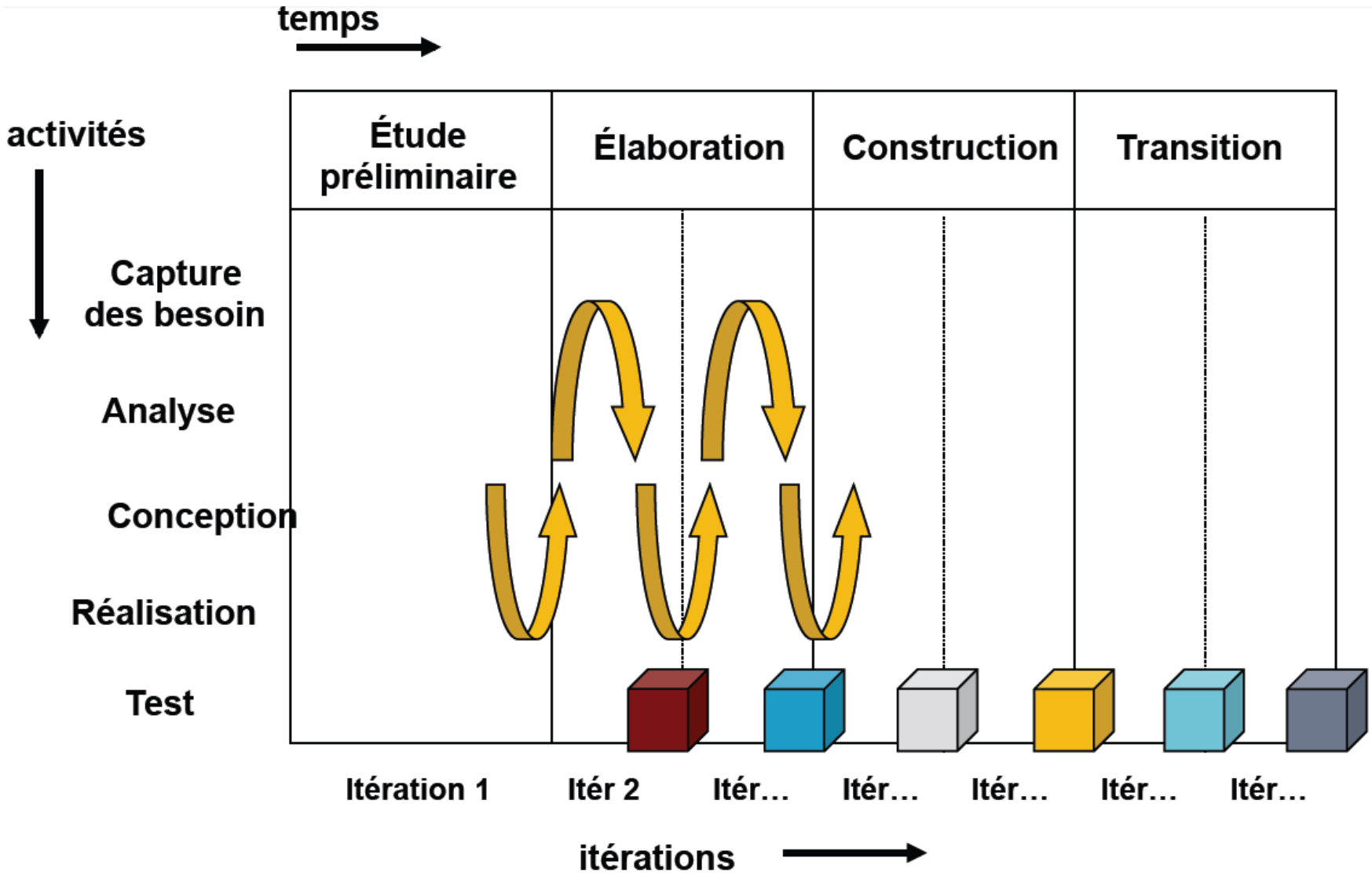
Des itérations

- 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
- Par les risques
- Par l'architecture
- ..

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

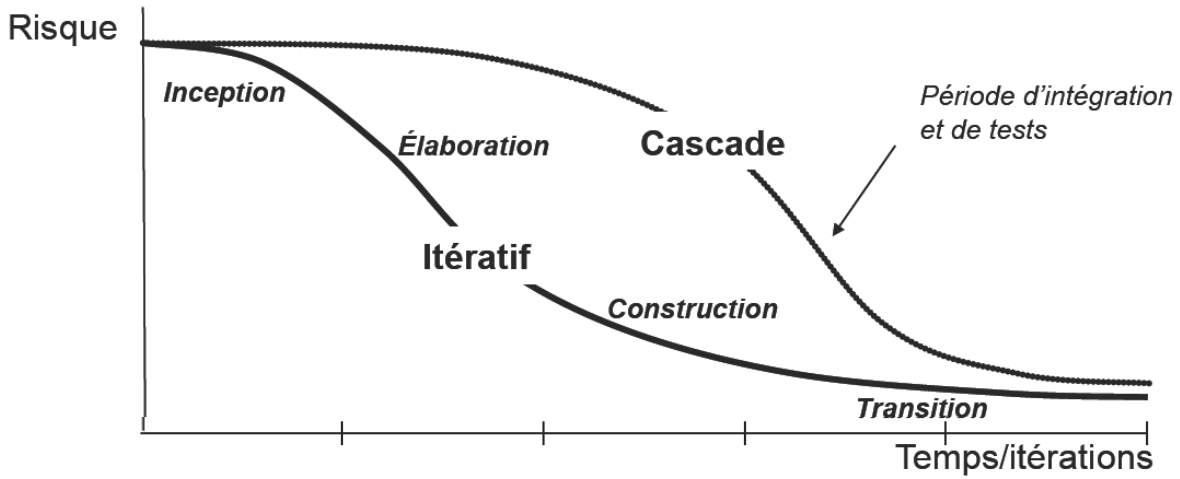
- Les cas d'utilisation spécifient le besoin à travers les objectifs utilisateurs

- 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

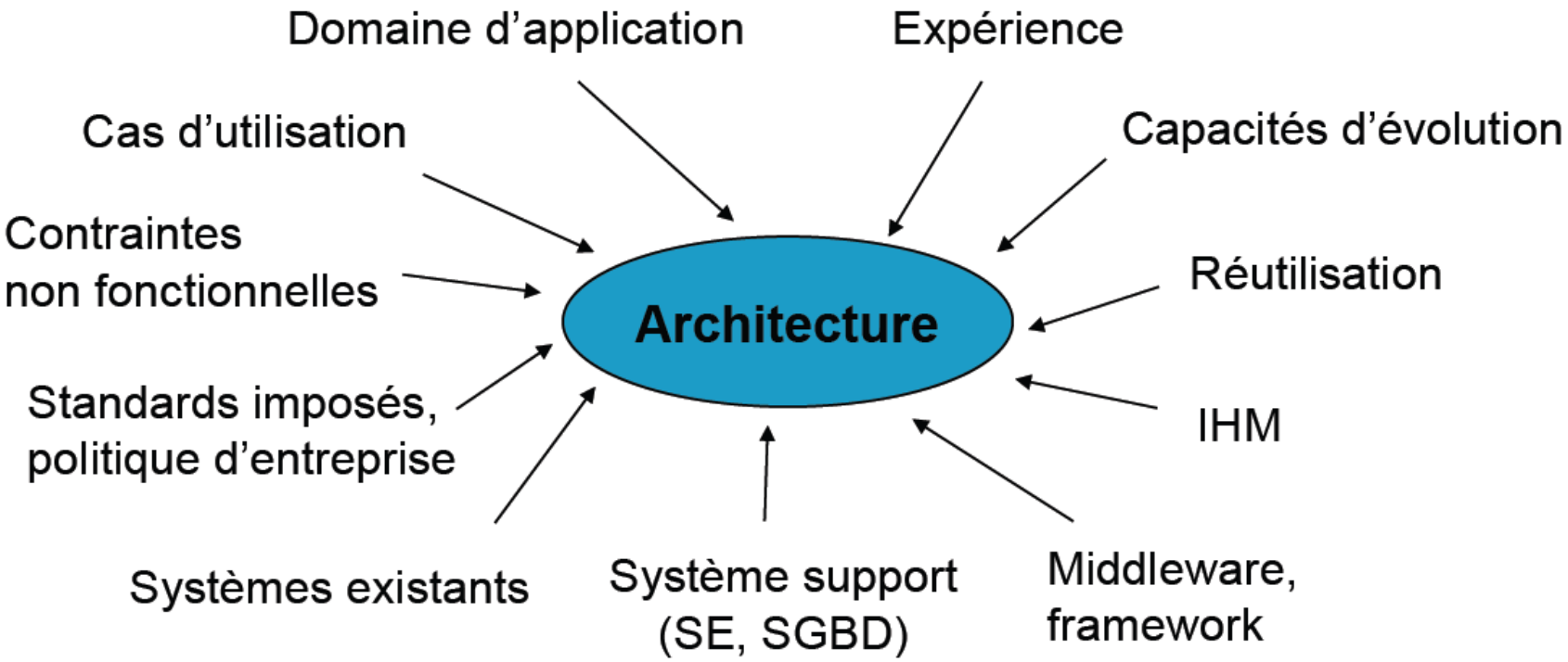
Architecture

- 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

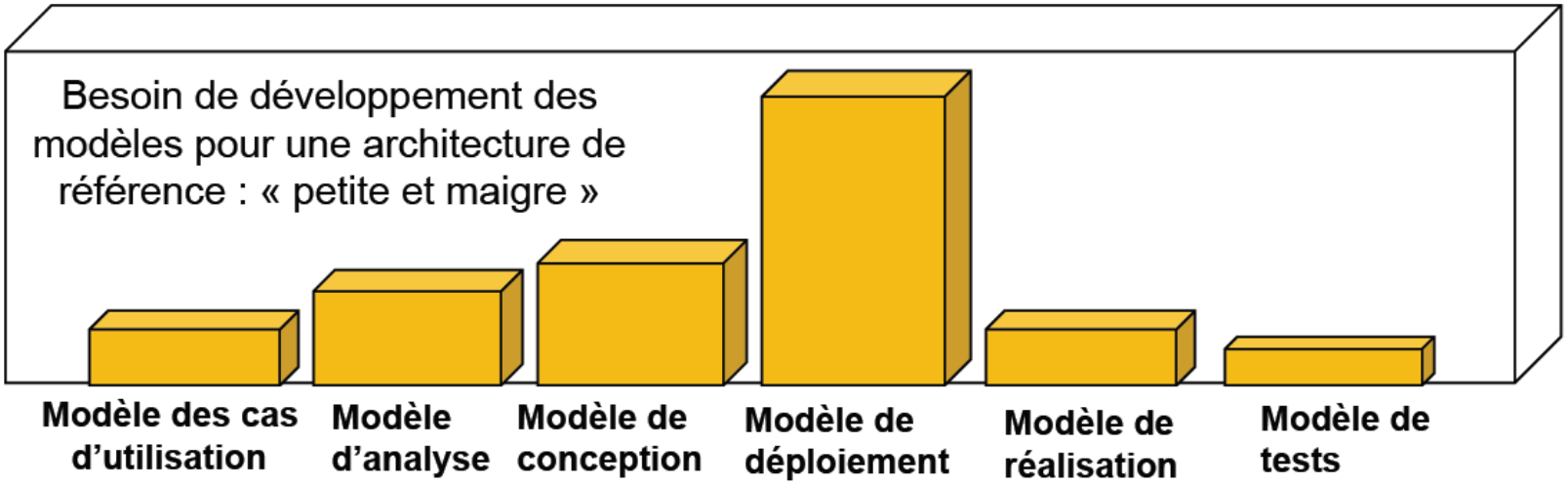
- **Démarrage, choix d'une architecture de haut-niveau et construction des parties générales de l'application**
  - ébauche à partir
    - de solutions existantes
    - de la compréhension du domaine
    - de parties générales aux applications du domaine (quasi-indépendant des CU)
  - des choix de déploiement

- **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

Activités pour passer des besoins au code

Activité : acquisition des besoins

Objectifs

- **Déterminer les valeurs attendues du nouveau système informatique**
- **Recenser les besoins**
  - les classer par priorité, évaluer leur risque
- **Comprendre le contexte**
  - Vocabulaire commun
  - Modèle du domaine = diagramme de classes
  - Modèle du métier (éventuellement) = UC + diag. d'activités

Modèle du domaine

- Des classes représentant des objets métiers et du monde réel
- Quelques attributs, peu d'opérations
- Des associations !
- Ce qui n'est pas encore modélisé va dans le glossaire (pour plus tard)

Modèle du métier (facultatif)

- Des liens entre des acteurs **(ce qu'on n'a pas le droit de faire d'habitude en UML)**
- Des diagrammes d'activités (des workflows)

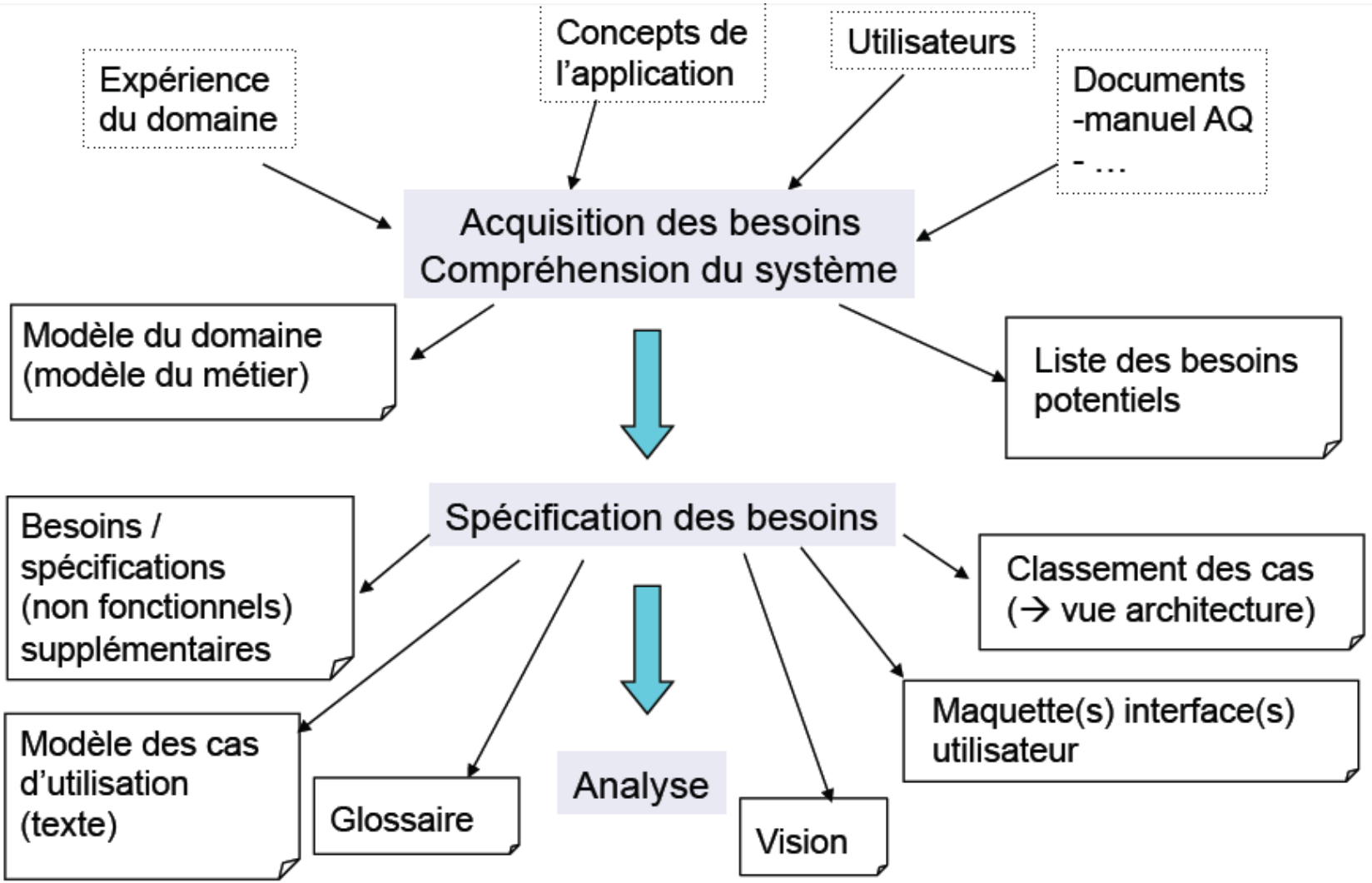
Activité : expression des besoins

Objectifs

- Définir les besoins fonctionnels
  - Scénarios
  - Acteurs / objectifs
  - Cas d'utilisation (donc construction du modèle de cas d'utilisation)
- Classer les cas d'utilisation par priorité
  - Fonction des risques, des nécessités de l'architecture ou du client
- Détailler les cas d'utilisation
  - **UC: Réserver une salle**
  - **Portée : système de planning**
  - **Acteur principal : secrétaire**
  - **Préconditions : une salle est libre pour la période désirée**
  - **Scénario nominal : …**
- Appréhender les besoins non fonctionnels
  - Contraintes sur le système
  - Environnement, plateforme, fiabilité, performance

Activité : expression des besoins

❒ Artefacts



Activité : expression des besoins

- **Travailleurs**

  - Analyste du domaine
    - Modèle du domaine
    - Modèle des cas d'utilisation
    - glossaire
  - Spécificateur de cas d'utilisation
    - Cas d'utilisation **détaillé**
  - Concepteur d'interface utilisateur
    - Maquette IHM
  - Architecte logiciel
    - Vue architecturale du modèle des cas d'utilisation

Activité : analyse

Objectifs

- Construire le modèle d'analyse et préparer la conception
  - Forme/Architecture générale du système : **recherche de stabilité**
  - Haut niveau d'abstraction

Mener l'analyse architecturale

- Identifier les packages d'analyse par regroupement logique
- Identifier les classes constituant le cœur de métier
  - 3 stéréotypes : frontière (interface), contrôle, entité
  - Les responsabilités doivent être évidentes
- Identifier les besoins non fonctionnels communs pour les rattacher aux UC

Analyser les cas d'utilisation

- Réaliser les scénarios
- Identifier les classes, attributs et associations nécessaires
- Décrire les interactions (typiquement par des diagrammes de séquence)

Activité : analyse

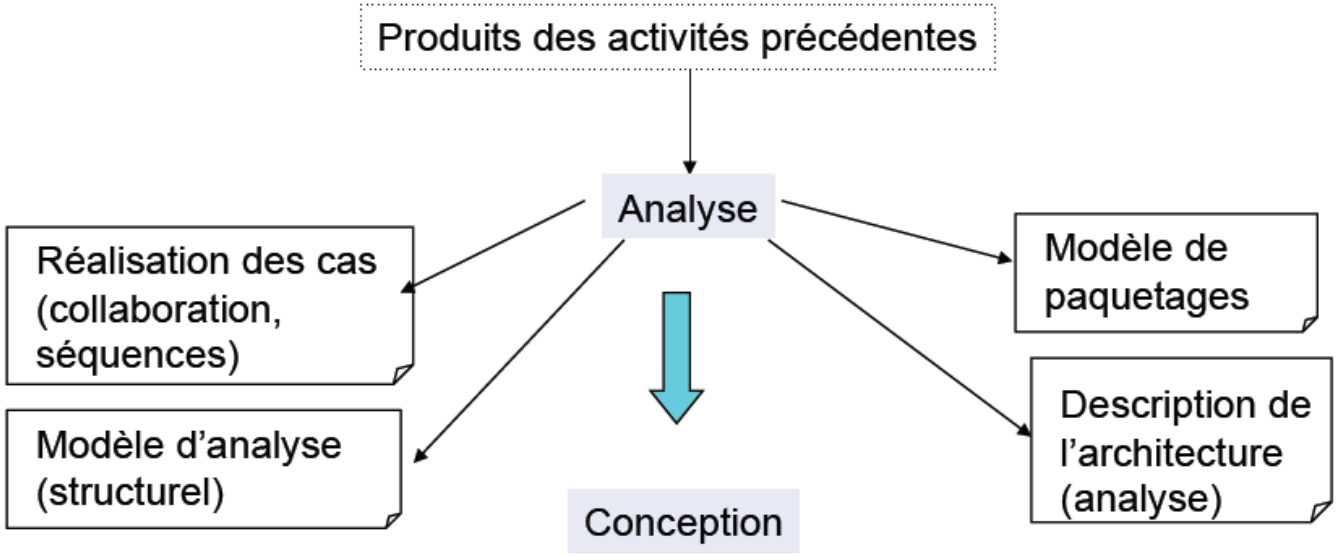
- **Le modèle structurel doit être l'union des associations déterminées**

- **Préciser les classes d'analyse**
  - Responsabilité ?
  - Identifier attributs, associations, héritages
  - Stabiliser les packages

- **Faire ces opérations pour chaque cas d'utilisation…**

Activité : analyse

Artefacts



Travailleurs

- Architecte
  - Responsable de l'intégrité du modèle d'analyse et de la description de l'architecture
- Ingénieur des UC
  - Responsable de l'analyse de chaque UC
- Ingénieur des composants
  - Responsable des classes et paquetages d'analyse

Activité : conception

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

- **Effectuer la conception architecturale**
  - identifier les noeuds et la configuration du réseau (déploiement)

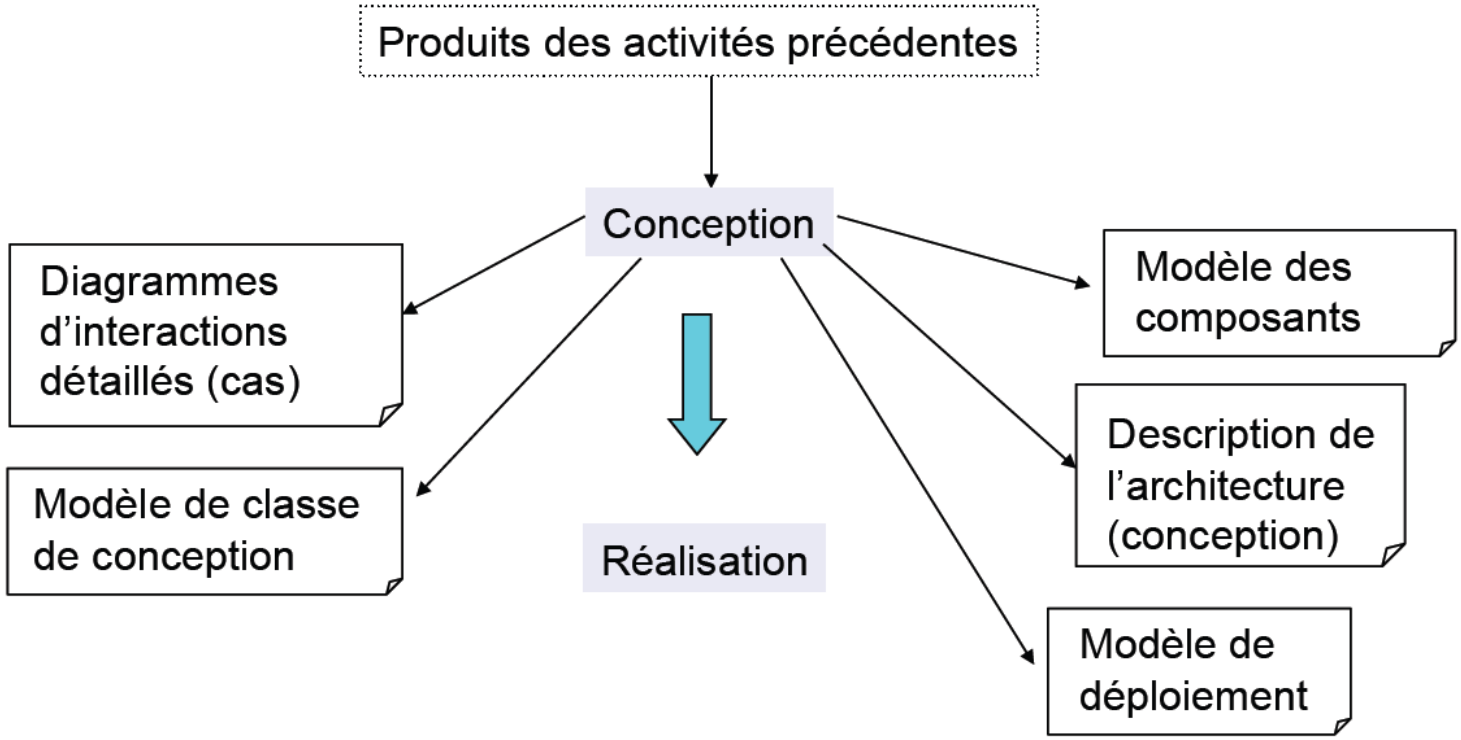
- **Concevoir les cas d'utilisation**
  - identifier les classes nécessaires à la réalisation des cas

- **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, les composants réseau et/ou middleware
  - permettra de piloter le travail des développeurs

Activité : conception

Artefacts



Travailleurs

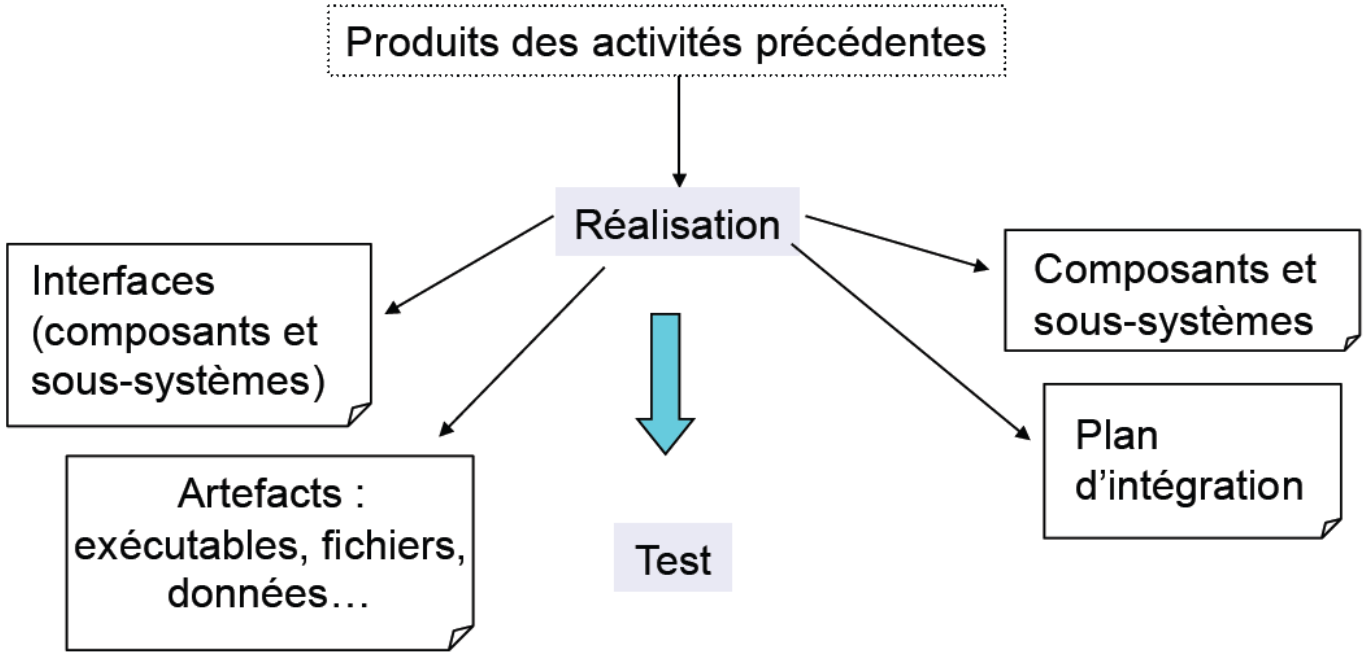
- Architecte
  - intégrité des modèles de conception et de déploiement
  - description de l'architecture
- Ingénieur UC
  - réalisation/conception des UC
- Ingénieur de composants
  - classes de conception, composants, interfaces

Activité : réalisation

- **Faire la mise en œuvre architecturale**
  - identifier les artefacts logiciels et les associer à des noeuds
- **Intégrer le système**
  - planifier l'intégration, intégrer les incréments réalisés
- **Réaliser les composants et les sous-systèmes**
- **Réaliser les classes**
- **Mener les tests unitaires**
    - tests de spécification en boîte noire (de structure en boîte blanche)

Activité : réalisation

Artefacts



Travailleurs

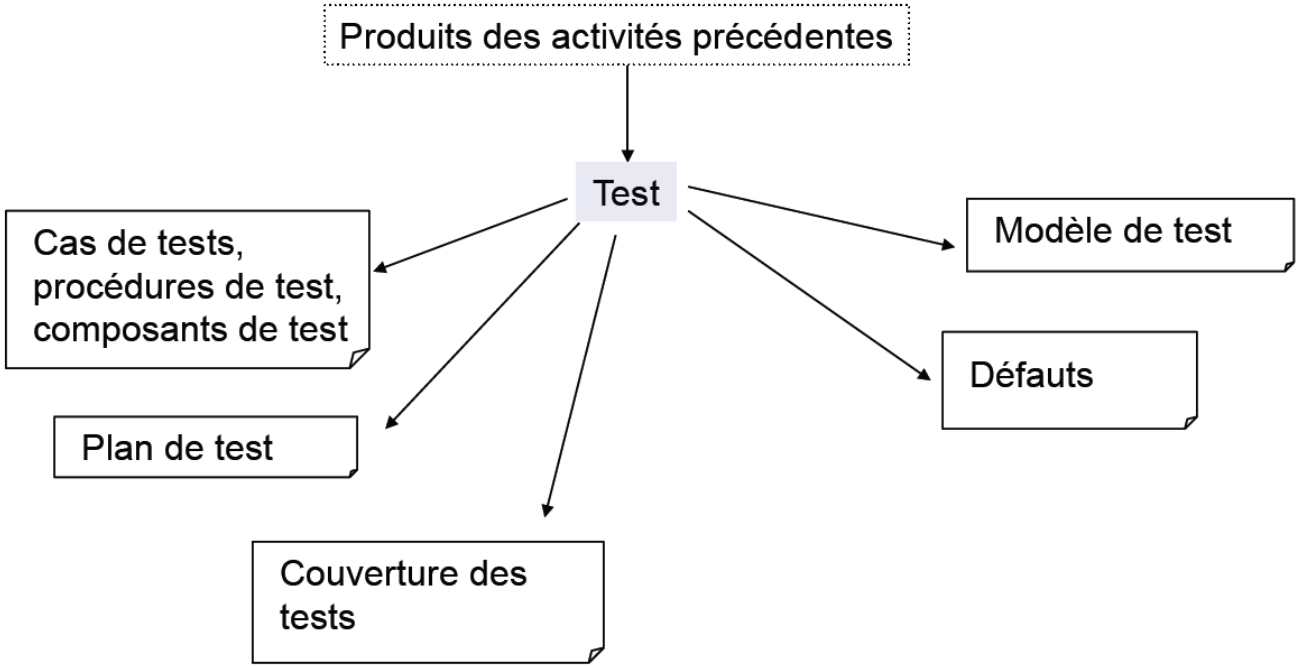
- Architecte
  - modèles d'implémentation et de déploiement
  - description de l'architecture
- Ingénieur de composants
  - artefacts logiciels, sous-systèmes et composants d'implémentation, interfaces
- Intégrateur système
  - plan de construction de l'intégration

Activité : test

- **Rédiger le plan de test**
  - Décrire les stratégies de test, estimer les besoins pour l'effort de test, planifier l'effort dans le temps
  - Tenir compte des risques (tester dès que possible)
- **Concevoir les tests**
- **Automatiser les tests**
- **Réaliser les tests d'intégration**
- **Réaliser les tests du système dans son intégralité**
- **Évaluer les tests**
  - Sont-ils efficaces ? Pertinents ?

Activité : test

Artefacts



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

Mini-conclusion

- UP
  - Est gros
  - Mais très structurant
- Il décrit un ensemble de processus applicables
- Mais il faut s'adapter aux besoins du projet
- Et on peut faire de l'agile sans faire de l'UP…

Méthodes « Agile »

Plan

- Principes
- eXtreme Programming

Genèse des méthodes Agile

- **Pas de méthode**
  - code and fix
  - Impossible sur les gros projets
- **Les méthodes monumentales**
  - méthodes, processus, contrats : rationalisation à tous les étages
  - problèmes et échecs
    - trop de choses sont faites qui ne sont pas directement liées au produit logiciel à construire
    - planification trop rigide
- **Années 90**
  - réaction à ces grosses méthodes
  - Puis pratique de consulting
  - Puis publication d'ouvrages
- **2001**
  - Agile manifesto
  - trouver un compromis : le minimum de méthode permettant de mener à bien les projets en restant agile
    - capacité de réponse rapide et souple au changement
    - orientation vers le code plutôt que la documentation

Principes

- **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

Principes et manifeste

- http://agilemanifesto.org/
- **Simplicité**
- **Légèreté**
- **Orientées participants plutôt que plan**
- **Pas de définition unique, mais un manifeste**
  - Février 2001
  - 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
    - a collaboration avec le client davantage que la négociation de contrat
    - la réponse au changement davantage que l'application d'un plan

Manifeste Agile : 12 principes

1. Our highest priority is to satisfy the costumer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face to face conversation.

Manifeste Agile : 12 principes

7. Working software is the primary measure of progress

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

9. Continuous attention to technical excellence and good design enhances agility

10. Simplicity – the art of maximizing the amount of work not done – is essential

11. The best architectures, requirements, and designs emerge from self-organizing teams

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

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

eXtreme Programming

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

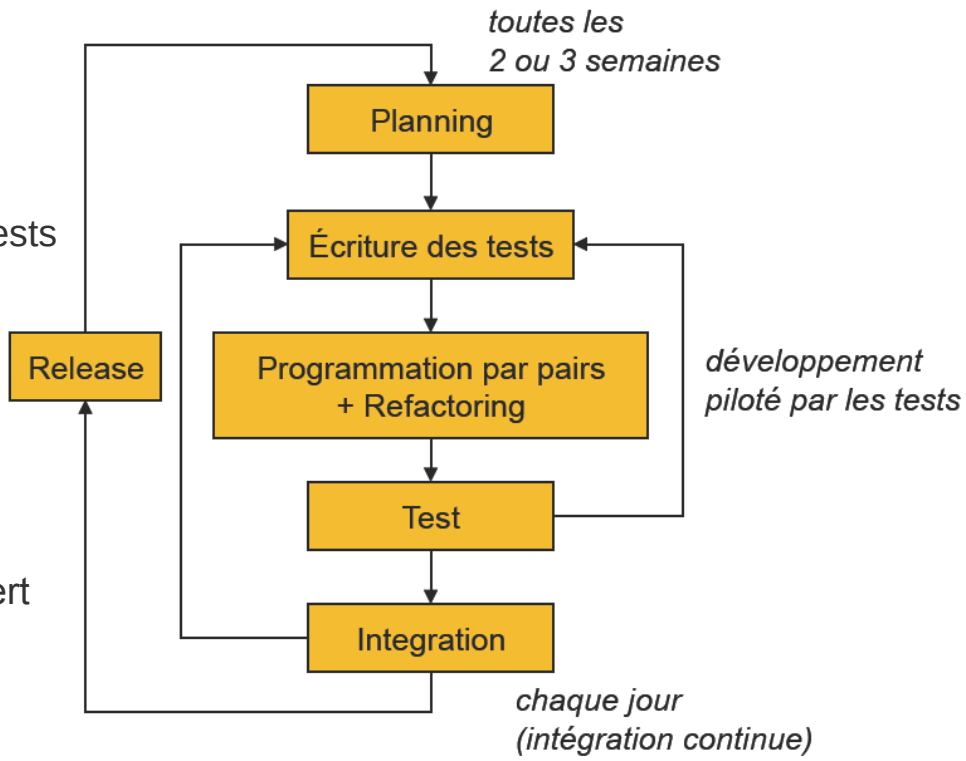
http://www.extremeprogramming.org

Déroulement global du projet

"A spike solution is a very simple program to explore potential solutions. Build the spike to only addresses the problem under examination and ignore all other concerns."

Déroulement et acteurs

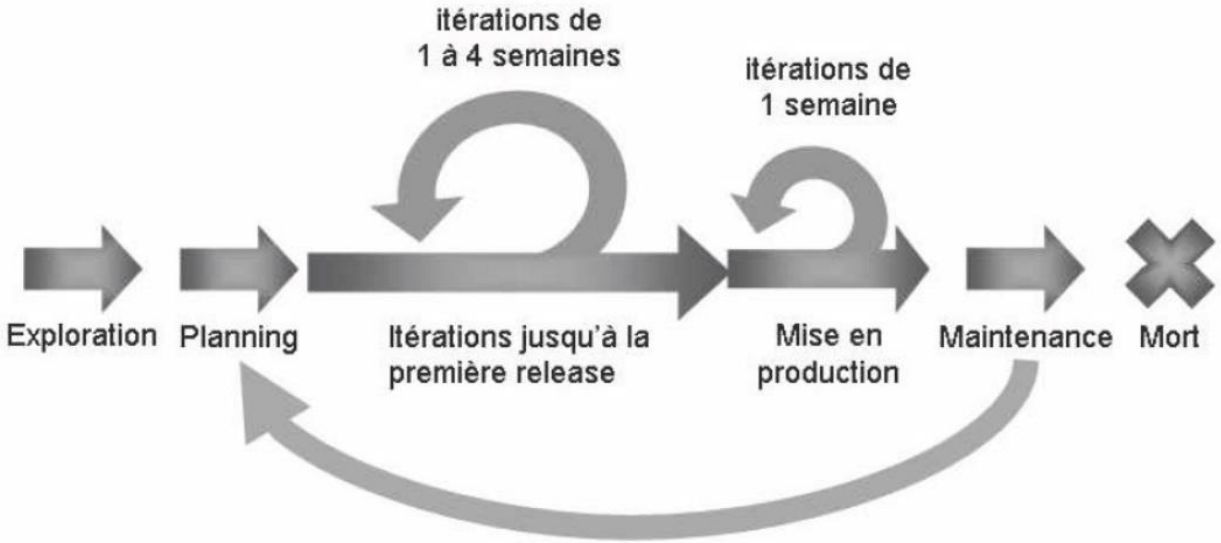
- **Client**
  - écrit les histoires et les tests fonctionnels
- **Testeur**
  - aide le client à écrire les tests, prépare les tests automatiques
- **Programmeur**
  - écrit les tests et puis code
- **Coach**
  - aide l'équipe par rapport au processus, expert méthode
- **Tracker**
  - suit les développement, vérifie que l'équipe ne perd pas la bonne direction
- **Manager**
  - Responsable infrastructure et soucis extérieurs
- **Consultant**
  - fournit les connaissances spécialisées au besoin



Pratique

- **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



Pratique

Métaphore

- chaque projet a une métaphore pour son organisation, qui fournit des conventions faciles à retenir

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

Pratique

Programmation par paires

- tout le code de production est écrit par deux programmeurs devant un ordinateur
- l'un pense à l'implémentation de la méthode courante, l'autre à tout le système
- les paires échangent les rôles, les participants des paires changent

Propriété collective du code

- tout programmeur qui voit une opportunité d'améliorer toute portion de code doit le faire, à n'importe quel moment

Intégration continue

- utilisation d'un gestionnaire de versions (e.g., SVN)
- tous les changements sont intégrés dans le code de base au minimum
- chaque jour : une construction complète (build) minimum par jour
- 100% des tests doivent passer avant et après l'intégration

Pratique

- **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
    - il ne devrait pas être possible de savoir qui a écrit quoi
- **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
    - par les résultats des tests

XP : bilan

Avantages

- Concept intégré et simples
- Pas trop de management
  - Pas de procédures complexes, ni de doc 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

Inconvénients

- Ne passe pas à l'échelle (pas plus de 10 développeurs)
- Risque d'avoir un code pas assez documenté
  - Difficile de faire reprendre le code par qqun en dehors de l'équipe
- Pas de conception générique
  - pas d'anticipation des développements futurs