Retours

- Retours sur COO
- Déroulements des TDs
- Canevas de Rapport

Université Nice SOPHIA ANTIPOLIS

Retour de COO : cours 2

- **Faux cas d'utilisation** (~~des interactions entre humains~~)
  - Hors interaction avec le système (dans un autre système)

- *Trop modéliser n'est pas modéliser juste. Bien délimiter le sujet, comprendre ce que l'on me demande et bien le faire.*

- **Les diagrammes de classe**
  - les acteurs apparaissent très souvent sous forme de classe !
  - *relation, classe, attributs, arités, opérations... construites au fur et à mesure.*

- **Les diagrammes de séquence**
  - Les paramètres qui tombent du ciel,
  - des séquences illogiques qui ne mènent à rien....

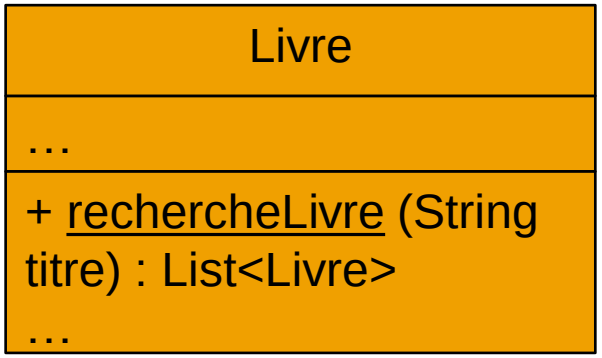
- ~~OCL : certains ont fait l'impasse, les autres ont réussis à condition de ne pas tomber dans le piège des collections !~~

- **Manque de cohérence entre les diagrammes...**

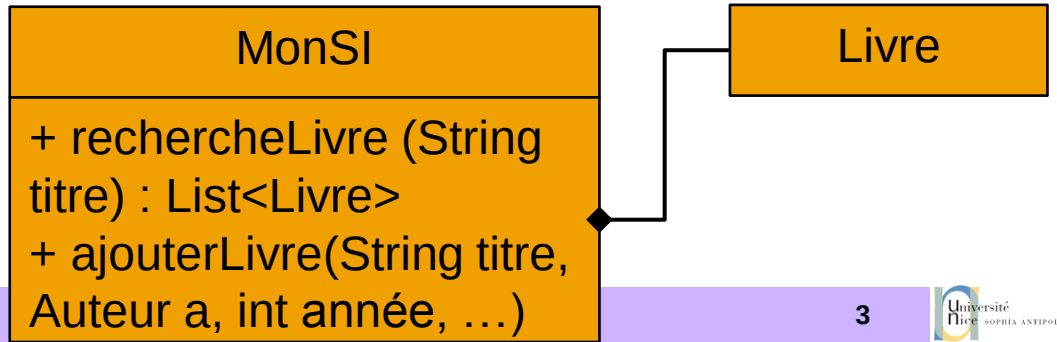
Retours généraux – rapport intermédiaire – en +

- des ids de partout : Objet vs BD
  - Manipulation d'un objet par son instance
  - L'objet a une « existance » propre
  - Id possible (sérialisation – référencement extérieur, recherche)

- Difficulté à retrouver les instances dans les collections
  - il faut préciser quand les instances sont « sauvegardées »
  - Listes statiques et méthodes statiques



  - classe "système" qui est une agrégation/composition de toutes les collections



Retours généraux – rapport intermédiaire – en +

- Différence entre le modèle et l'exemplaire (comme livre et exemplaire livre) => impact la recherche, la commande, etc.

- un diagramme d'état = états « uniques »
  - ne convient pas pour une réservation => une fois réservé pour une date, l'entité n'est plus disponible
  - Soit livre, un livre disponible
    Réservation du livre pour le 31 décembre => livre.setDispo(false)
    une réservation du même livre pour le 20 avril ? livre.isDispo( ) -> false
    on ne peut plus le réserver…
  - Solution… une liste de date : si la date est dans la liste, ce n'est pas disponible

- Ne pas confondre U.C. et activité
  - U.C. => unité de temps
  - liaisons entre U.C. = extends, includes, spécialisation
  - Pour des enchainements d'U.C. => diagramme d'activité
    Pour les transmissions de données, visible via séquence et IHM

Liaison IHM – U.C. – Séquences

- Mieux exploiter les IHM
- Utilisation pour faire le lien entre les U.C. et les séquences
  - Paramètres (à saisir)
  - Données à afficher
  - Nature de l'interaction (synchrone/ asynchrone…)
- Cohérence avec le diagramme de classe
  - Comment représenter les données…
  - informations que les classes doivent contenir
- Cohérence avec l'état
  - Représentation de l'état
- Cohérence avec l'activité
  - Enchainement d'IHM

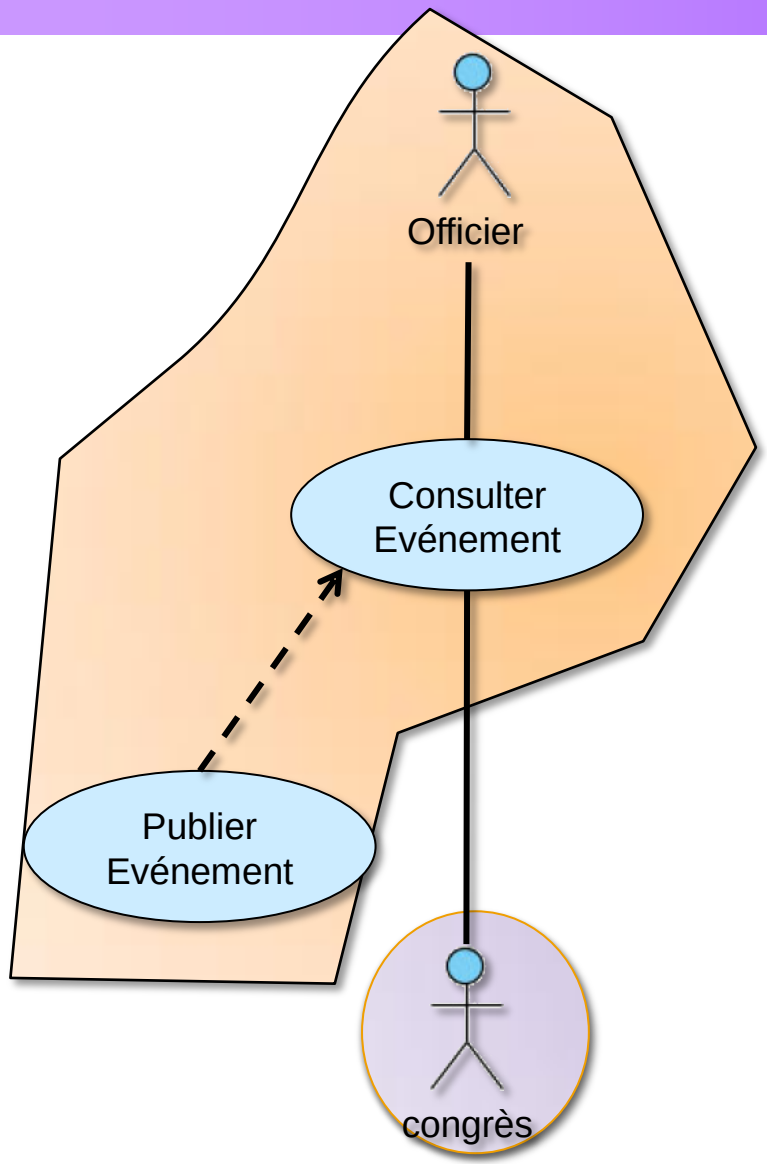
Rapport final

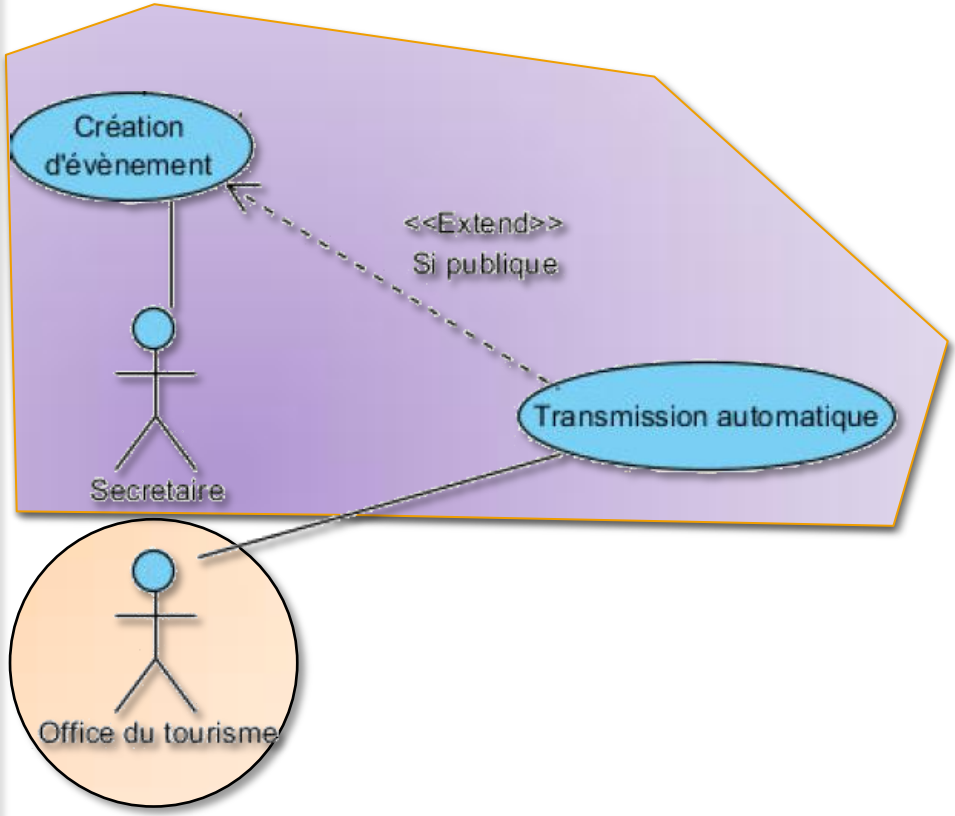
- **Contrôle Continu et note(s) différenciée(s)**
- **Règle du 20-60-20**
  - **Modalité prévue : 40% rapport intermédiaire, 40% rapport final, 20% soutenance**
  - **Bouée de sauvetage : si la progression est bonne, 20% rapport intermédiaire, 60% rapport final, 20% soutenance**
- **Critères n°1 :**
  - **La méthode**
  - **La cohérence**
    - **Interne**
    - **entre les groupes**
  - **La crédibilité**
- Collaboration : Préparation de la séance

Collaboration entre S.I.

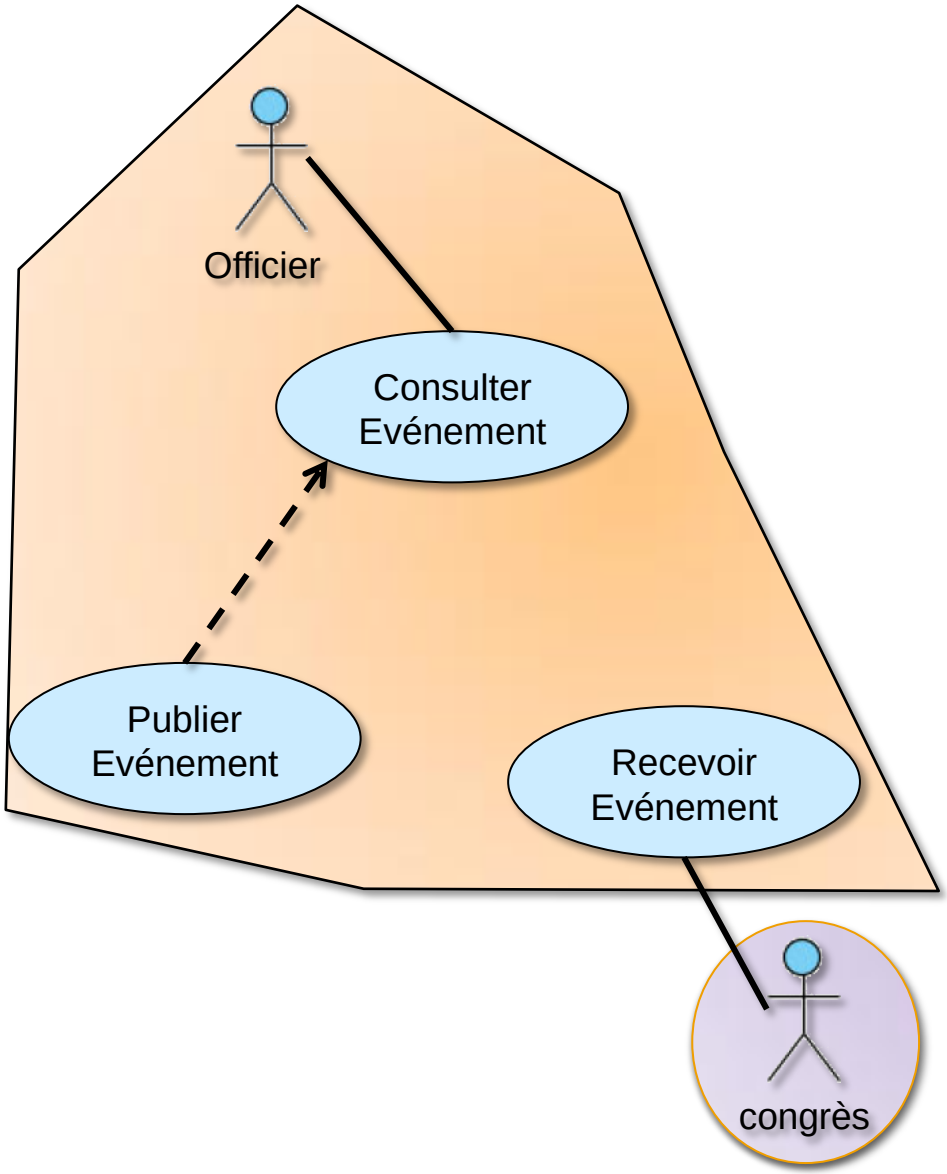
- Use Cases & scénario
- Séquences complémentaires
- Classes « compatibles » mais pas nécessairement les mêmes
- Cohérences dans la transmission
  - Informations échangées
  - média

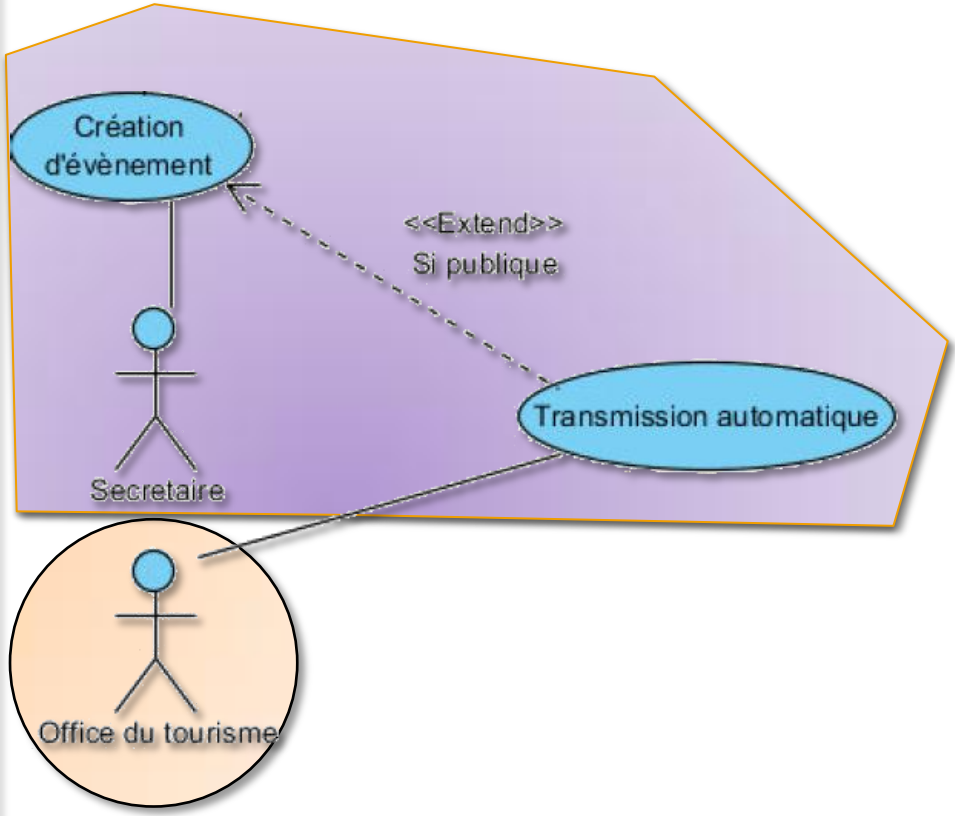
Correspondance de Use Case : version « synchrone »



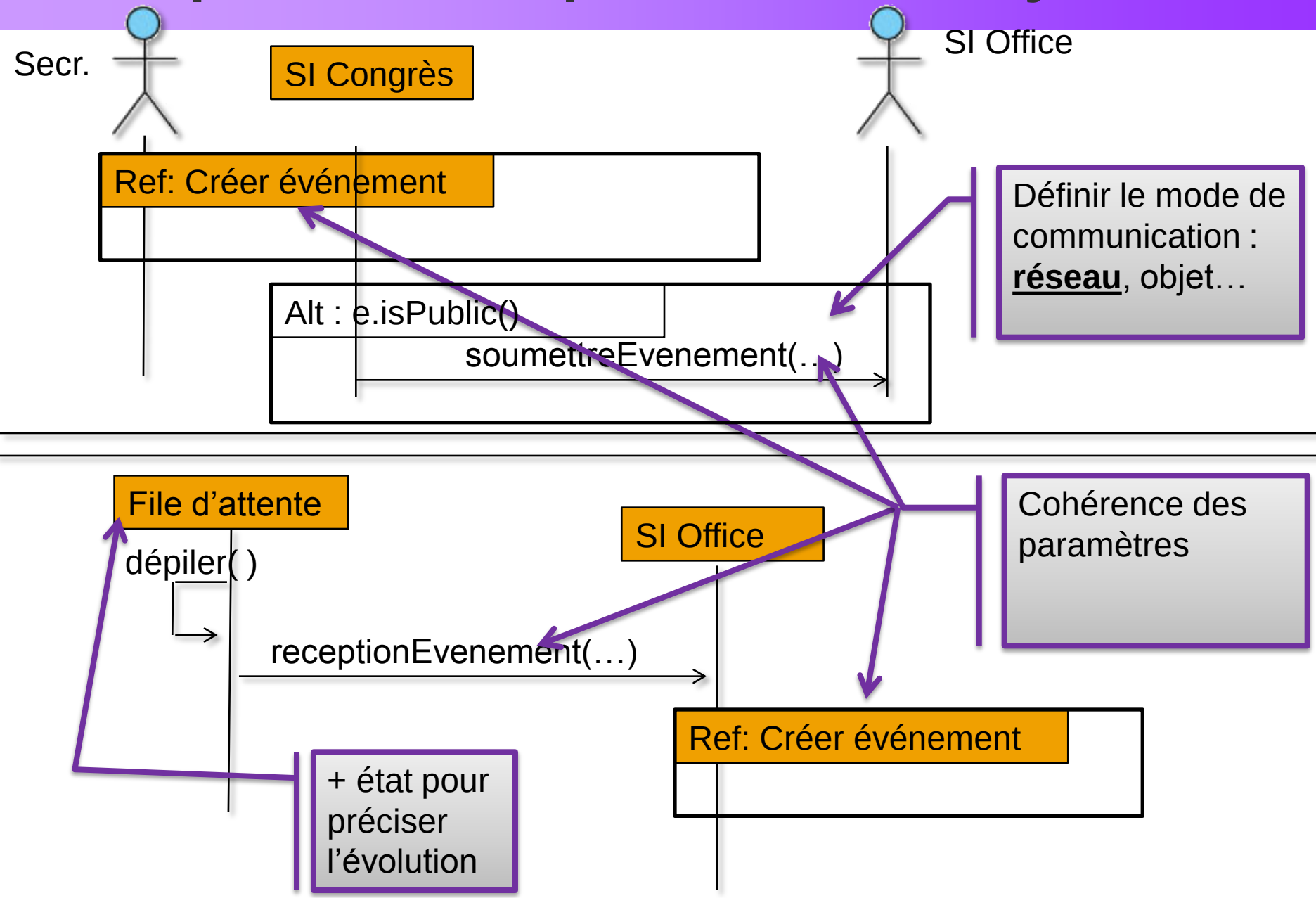


Correspondance de Use Case : version « asynchrone »

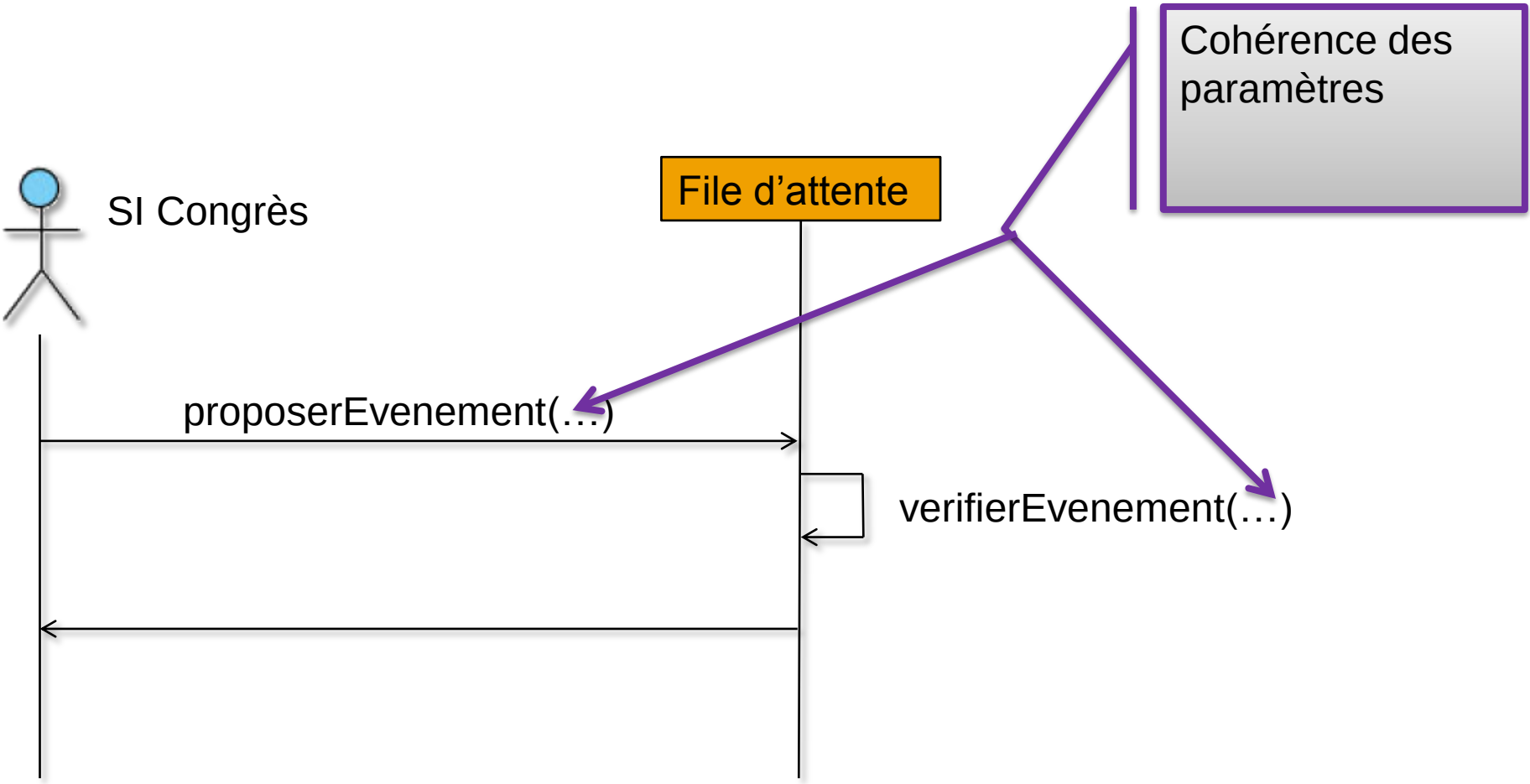




Correspondance de Séquences version « asynchrone »



Correspondance de Séquences version « asynchrone »



Canevas de rapport (1/3)

Pour la suite : TOUT DIAGRAMME DOIT ETRE EXPLIQUE

Mettre en valeur ce qui a changé : corrigé new

1. Introduction

- Résumé du sujet
- Résumé des points en interaction avec les autres équipes
- Problématiques à soulever
- **Ce qui a changer depuis le rapport intermédiaire** new

2. Point de vue général de l'architecture

- **Un glossaire**
- **Une représentation générale (diagramme d'activité)**

Canevas de rapport (2/3)

3. Point de Vue Statique

- Cas d'utilisation
  - Acteurs
  - Diagrammes de Cas d'utilisation
  - Scénarios (sous forme textuelle, un par *use case*)
  - Un morceau de Maquette IHM pour chaque use case
- Diagrammes de Classes lisible en format A4 (une découpe « logique » avec répétition de certaine classe).

4. Point de vue Dynamique

- Diagrammes de Séquence
- Chaque diagramme de séquence devra référencer un use-case.
- Morceau(x) d'IHM associé(s)
- Diagrammes de Machine d'Etat

Canevas de rapport (3/3)

5. **Interactions avec les autres S.I.**

   Ici ou dans les points 3 ou 4

corrigé

6. **Contraintes**

new

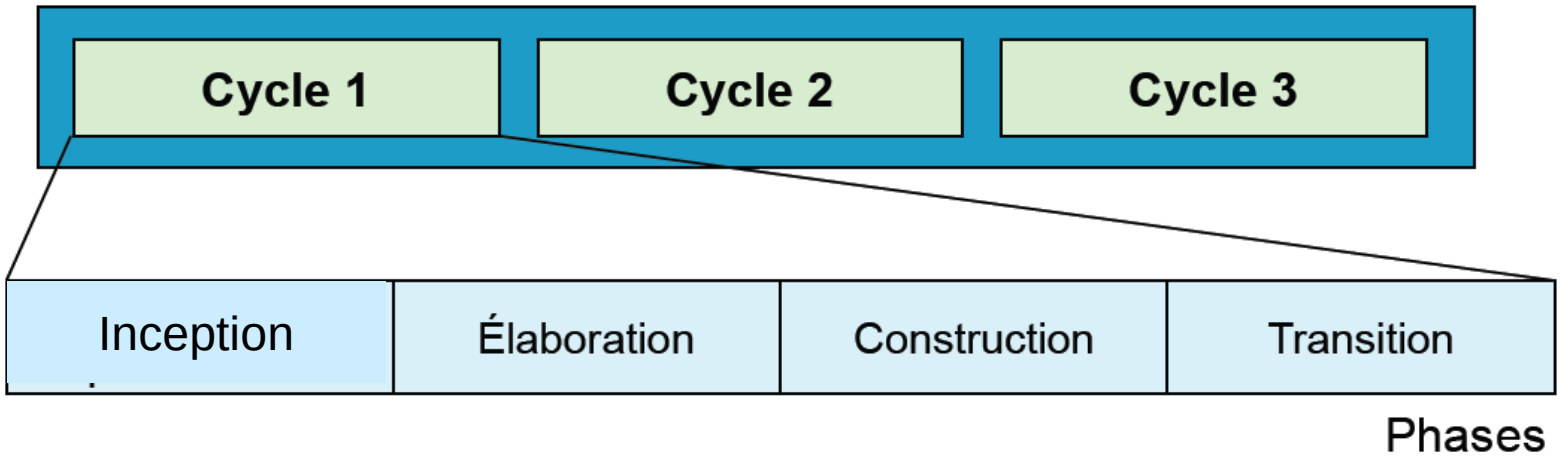
7. **Maquette de l'interface**
   - une maquette globale rattachée aux diagrammes présentés dans le document (synthèse des points 3-4-5)
   - Lien avec les scénarios

8. **Conclusion**
   - Analyse de votre solution : points forts et points faibles

Rappel sur UDSP

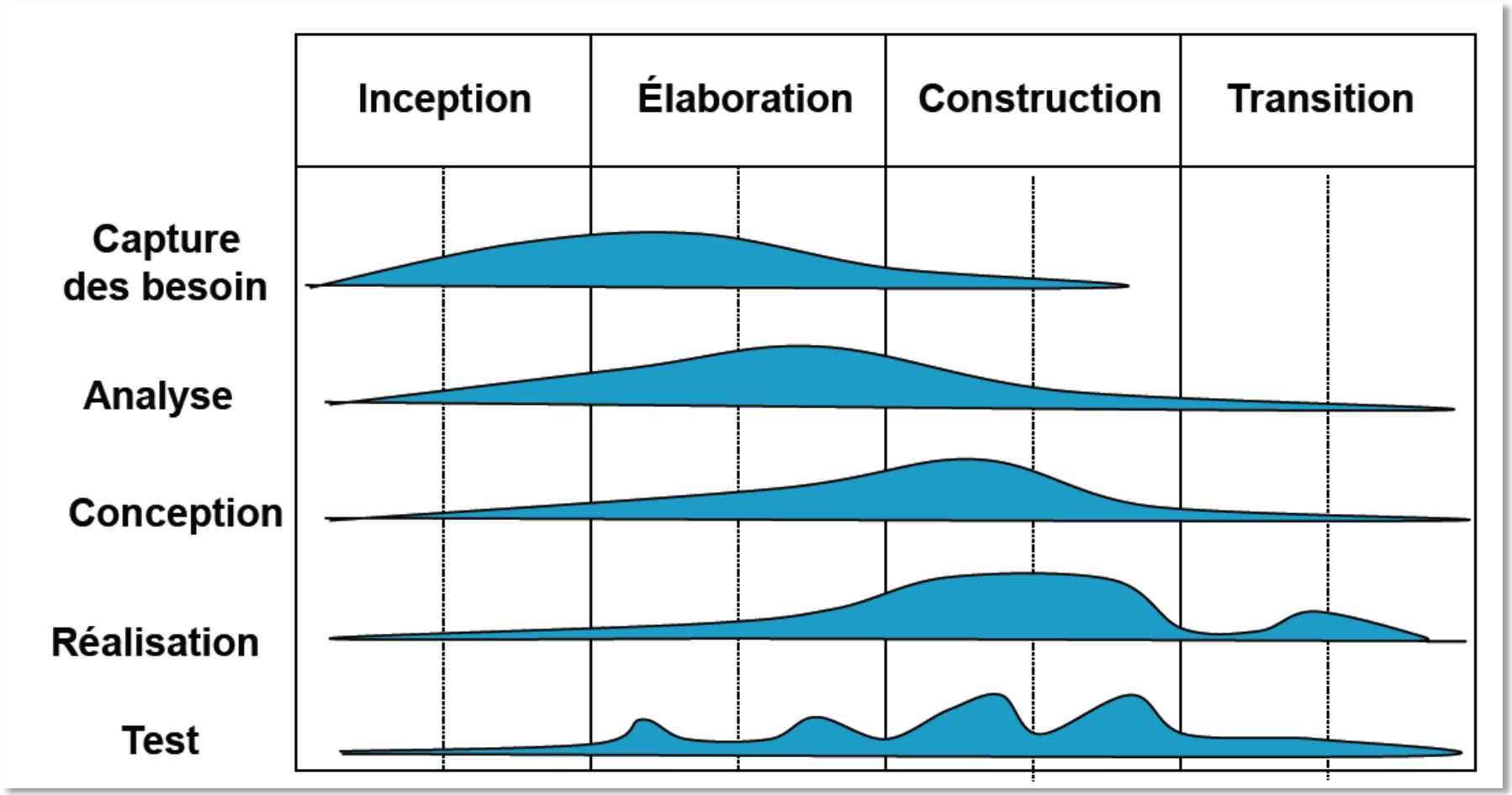
USDP : Principes

- 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

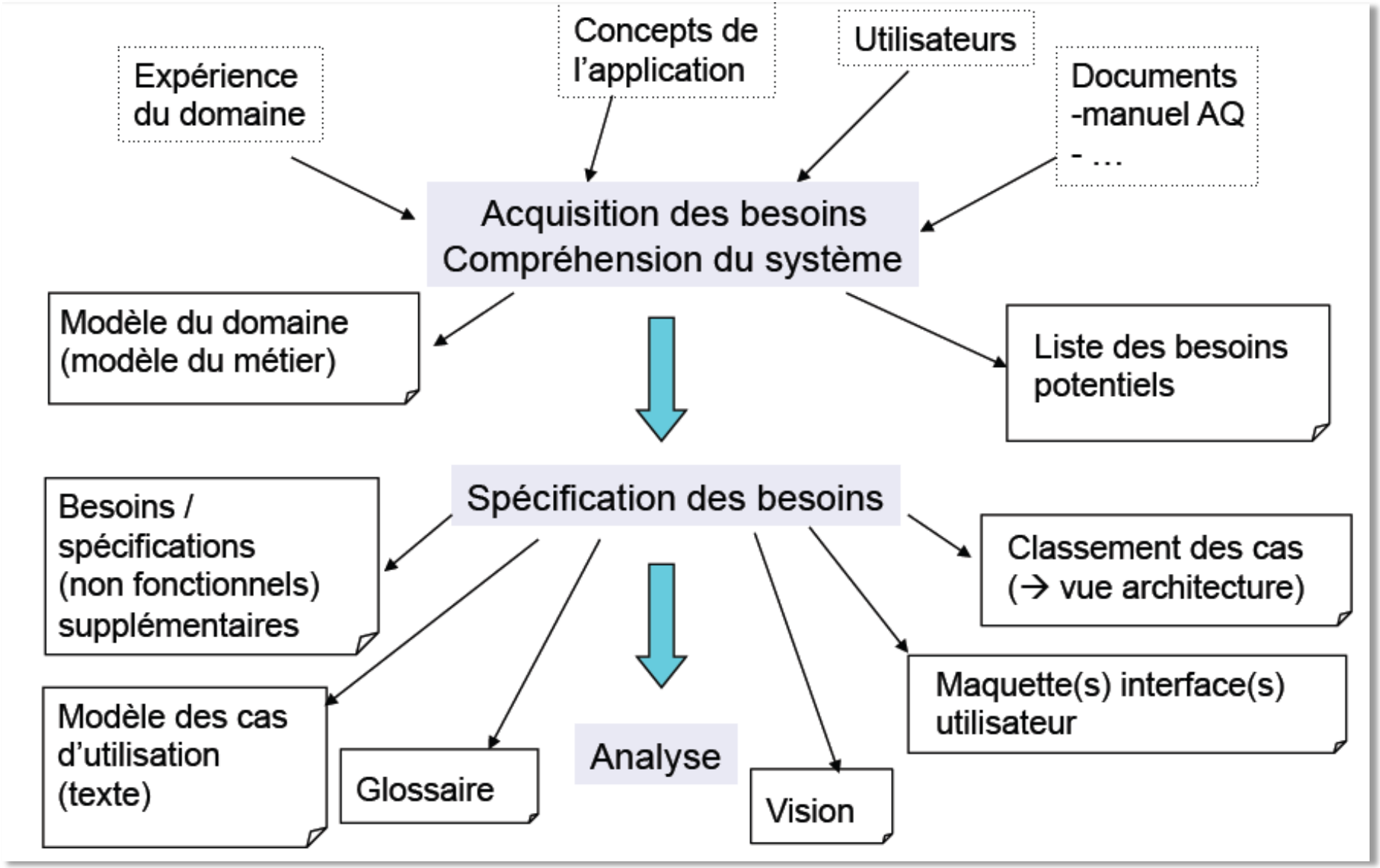


Effort lors des phases

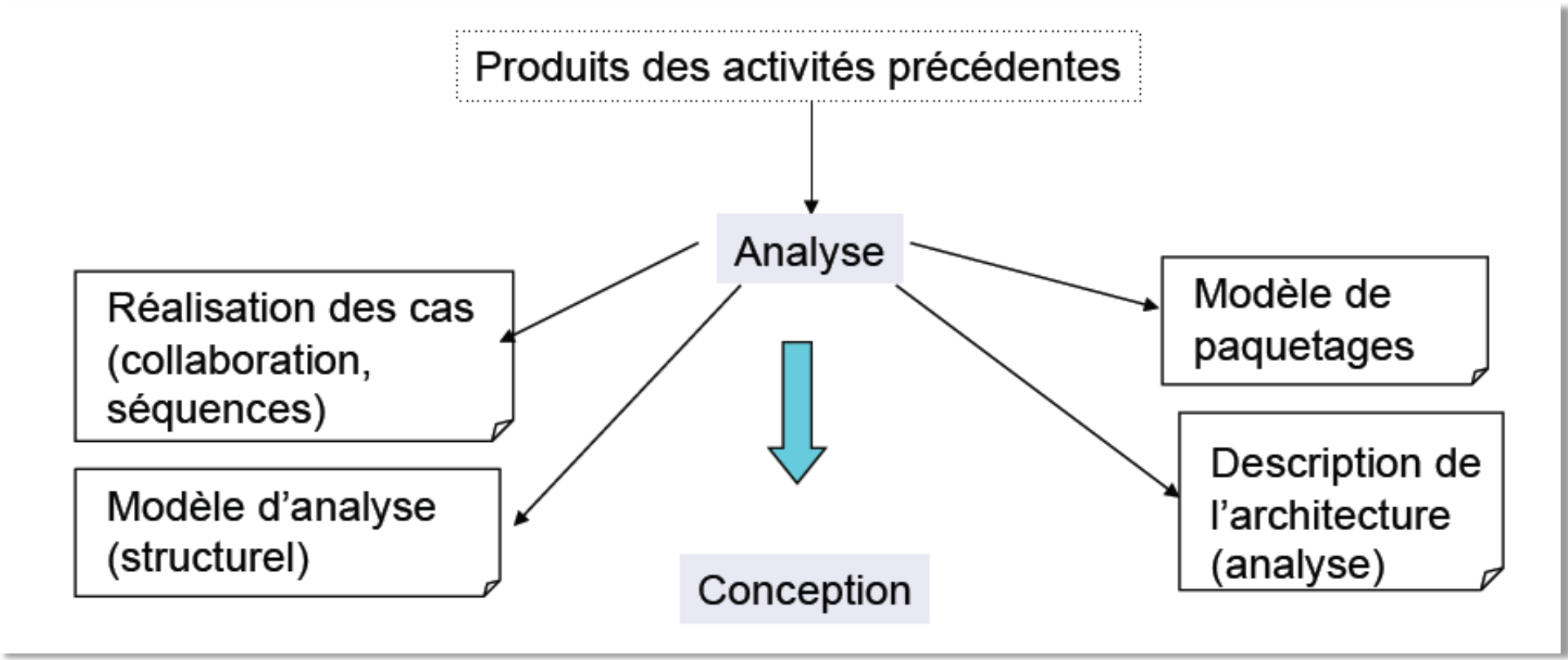
- Chaque phase spécifie les activités à effectuer



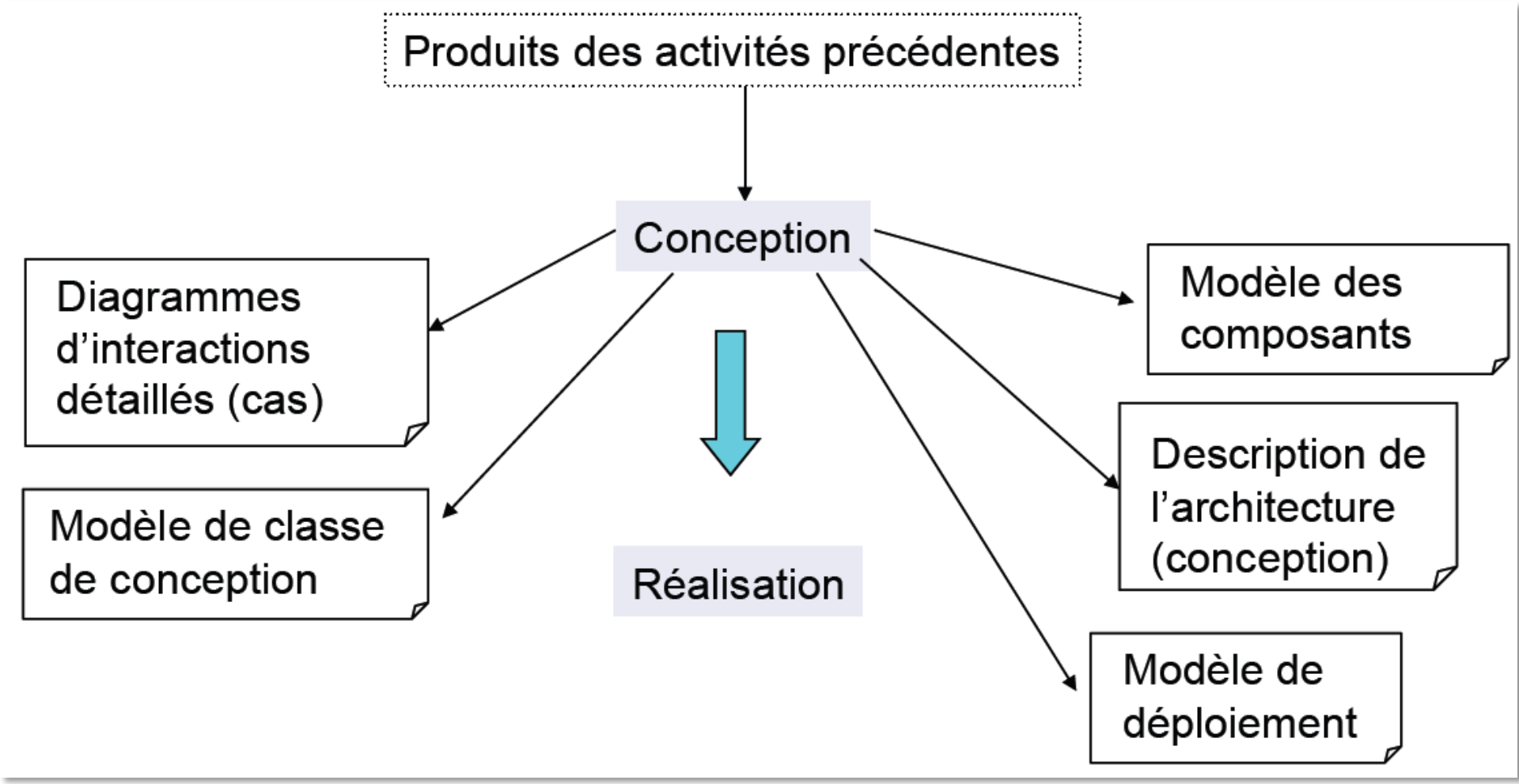
Productions



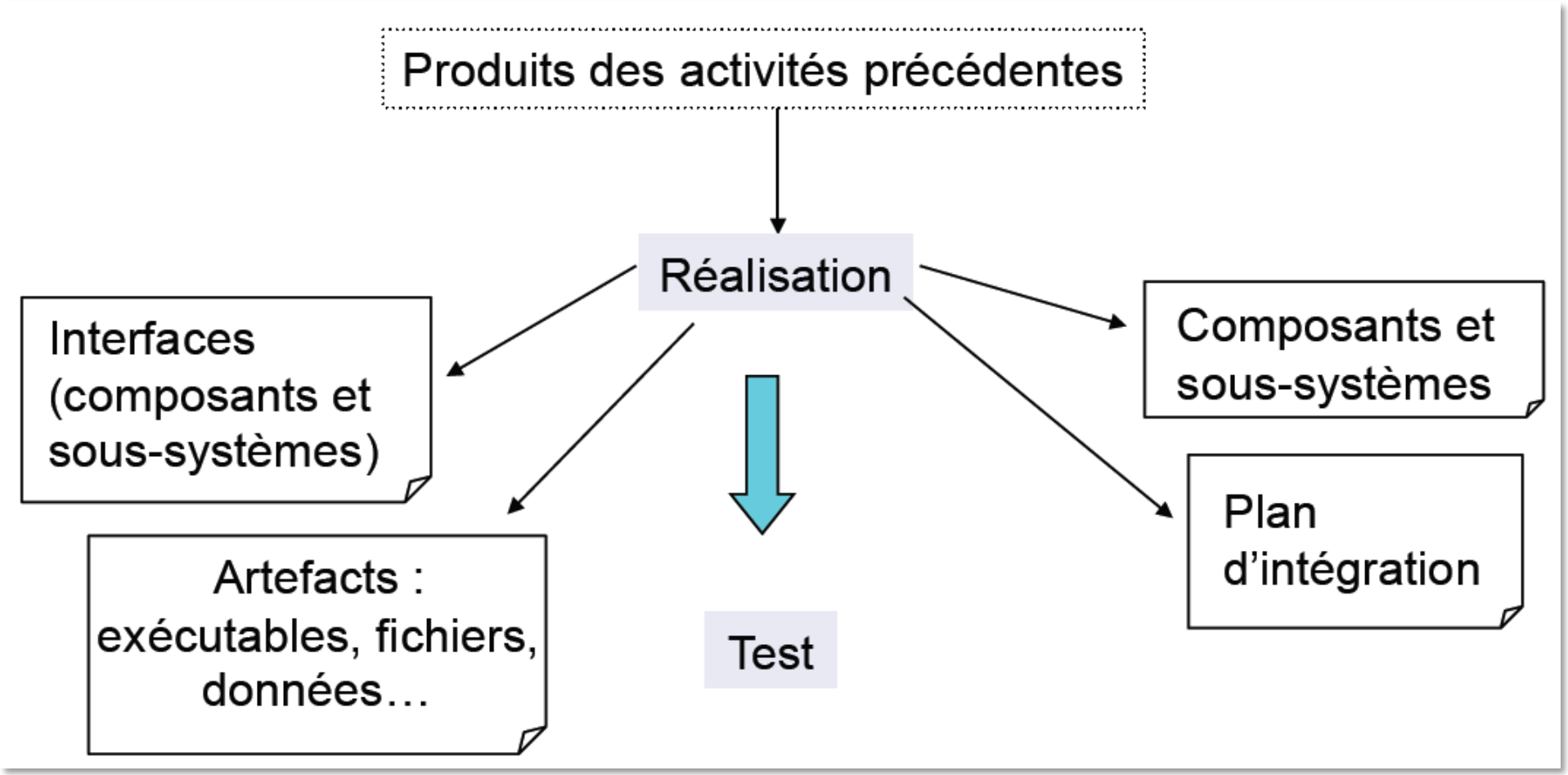
Productions



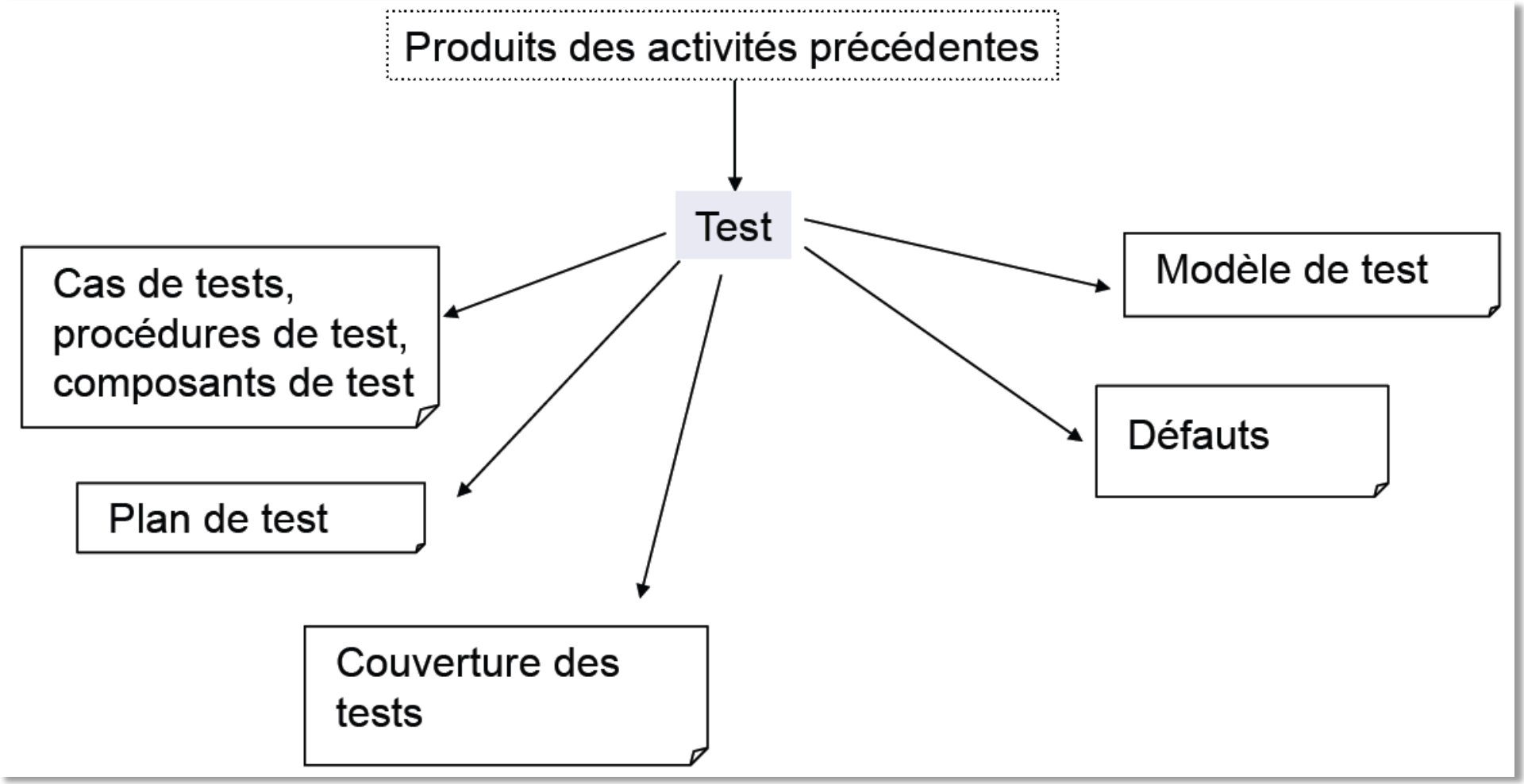
Productions



Productions



Productions

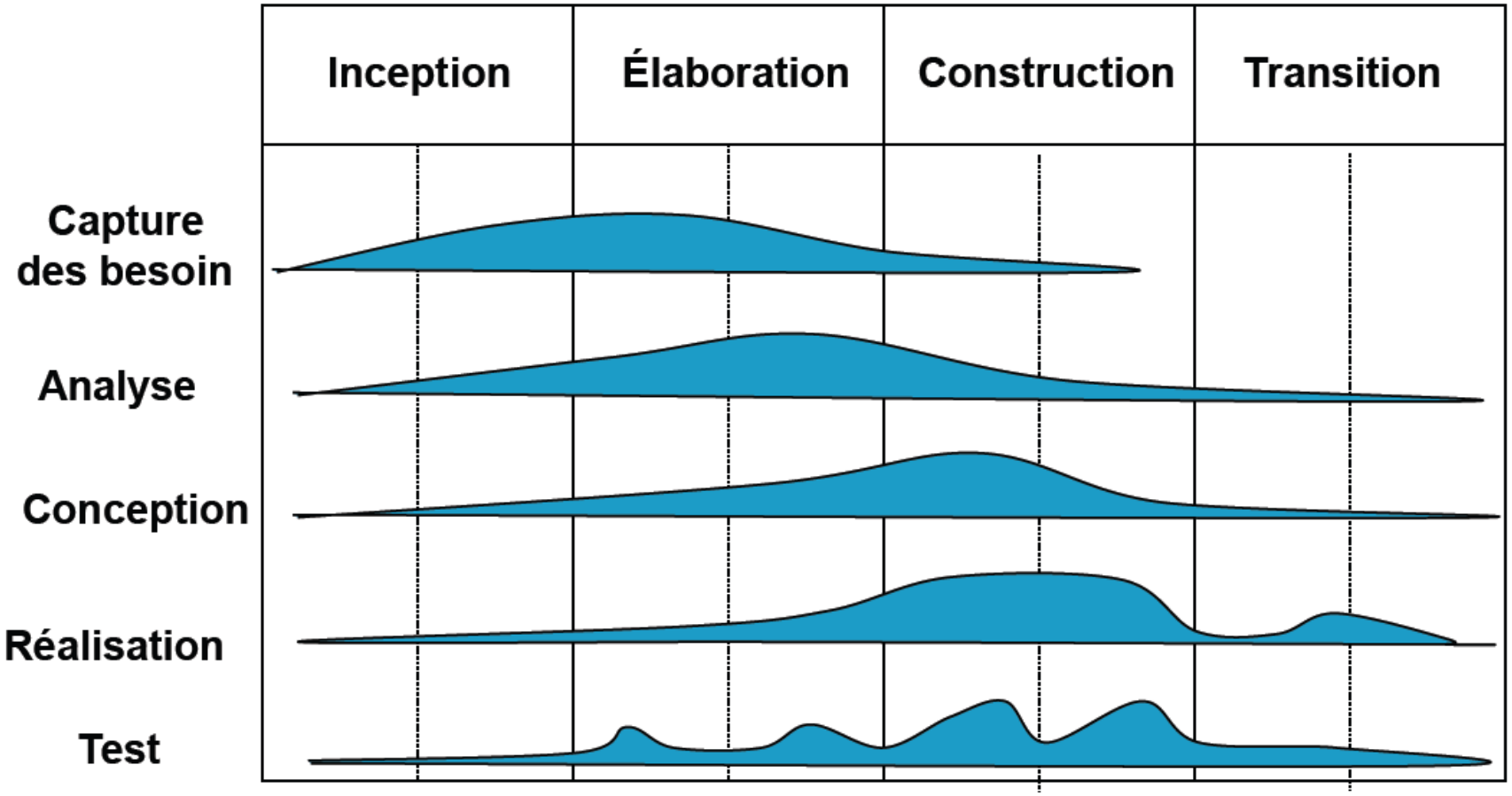


Phases de pilotage des activités

Université Nice SOPHIA ANTIPOLIS

Rappel sur les phases

- Chaque phase spécifie les activités à effectuer

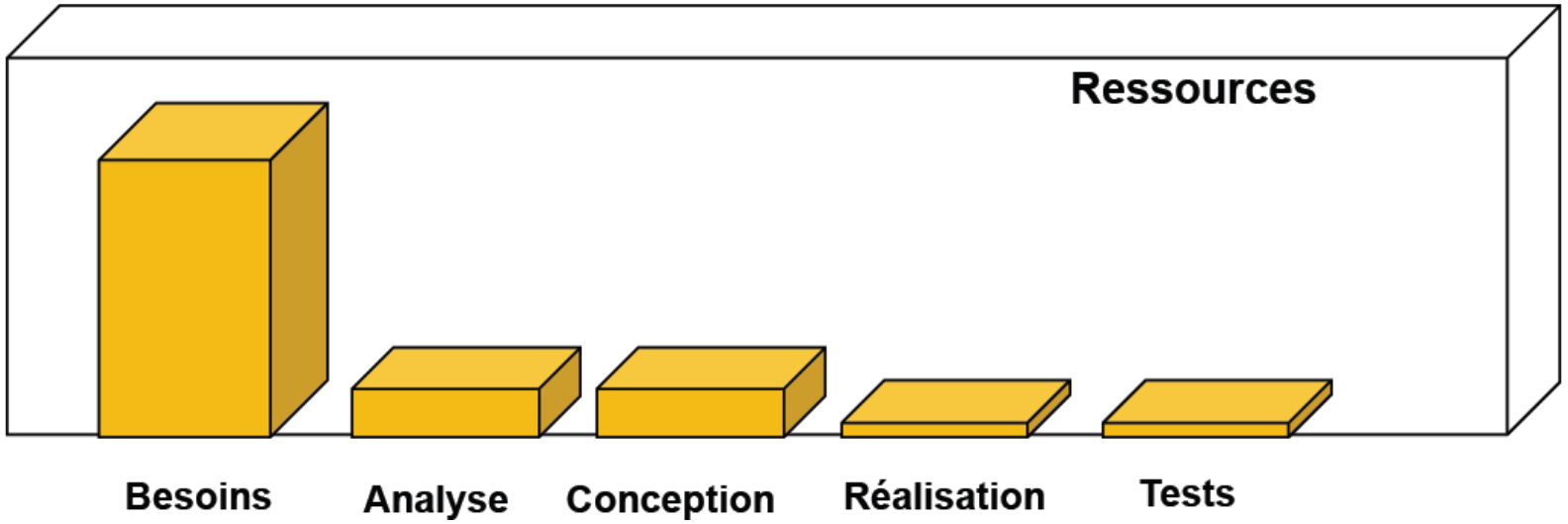


Gestion des phases

- **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

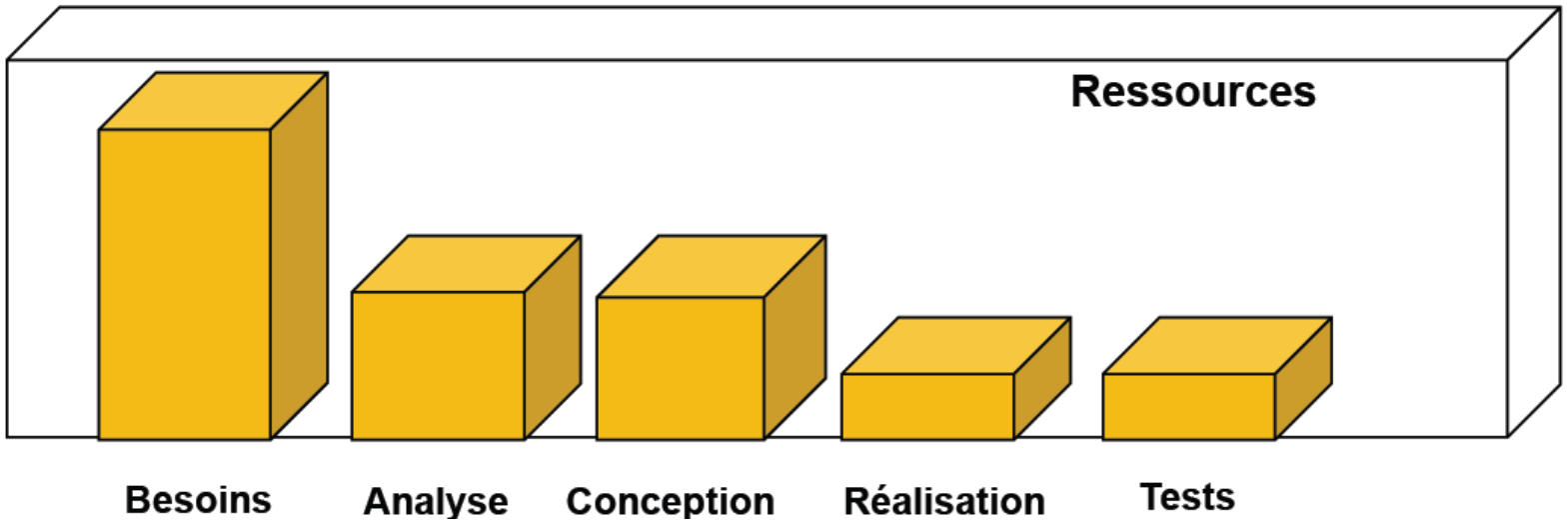
- **Capture des besoins**
  - comprendre le contexte du système (50 à 70% du contexte)
  - établir les besoins fonctionnels et non fonctionnels (80%)
  - traduire les besoins fonctionnels en cas d'utilisation (50%)
  - détailler les premiers cas par ordre de priorité (10% max)
- **Analyse**
  - analyse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

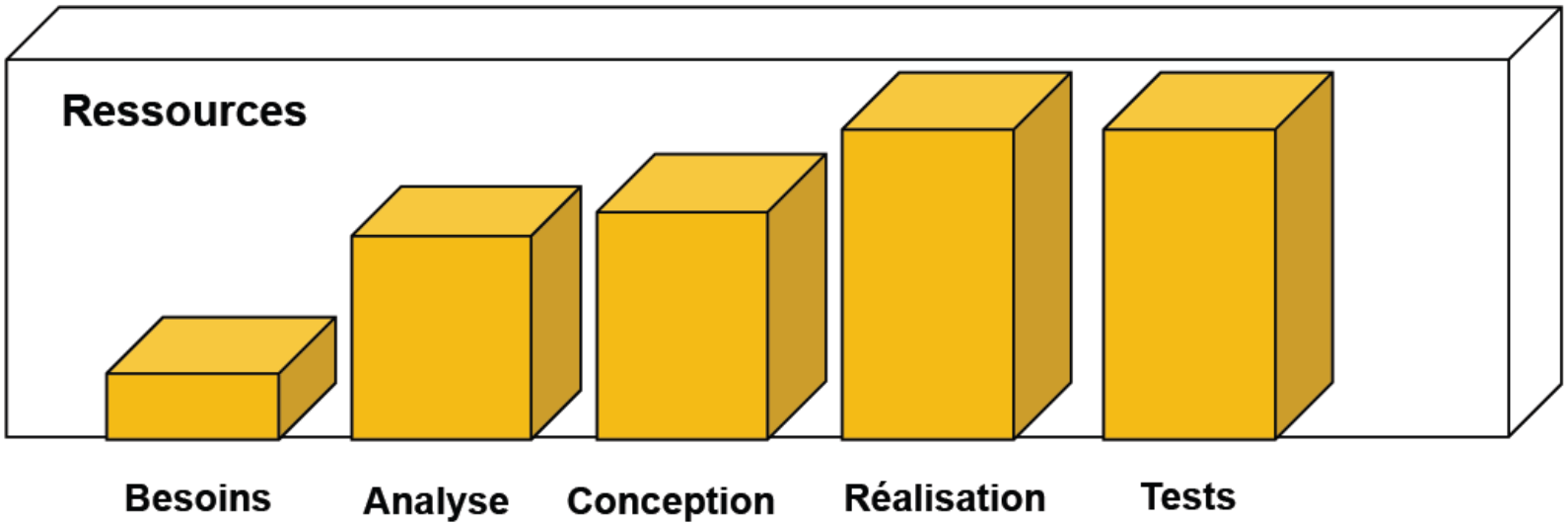
- **Capture des besoins**
  - terminer la capture des besoins et en détailler de 40 à 80%
  - faire un prototype de l'interface utilisateur (éventuellement)
- **Analyse**
  - analyse architecturale complète (packages...)
  - raffinement des cas d'utilisation (pour l'architecture, < 10%)
- **Conception**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

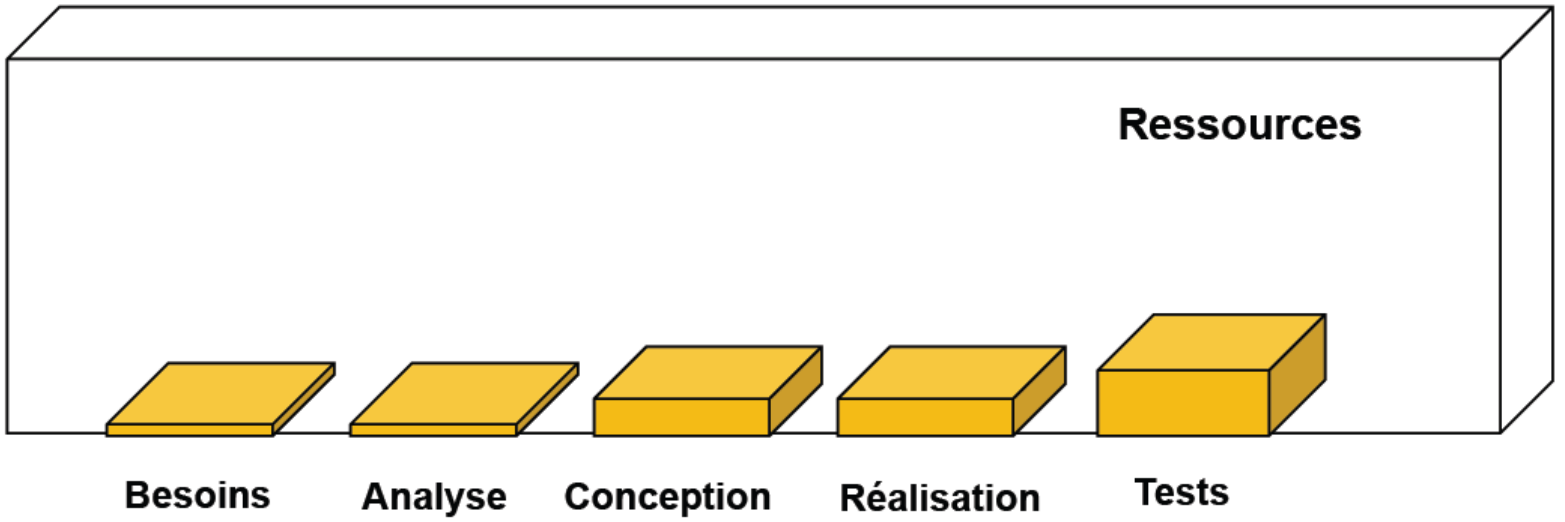


Activités et livrables de la construction

- **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



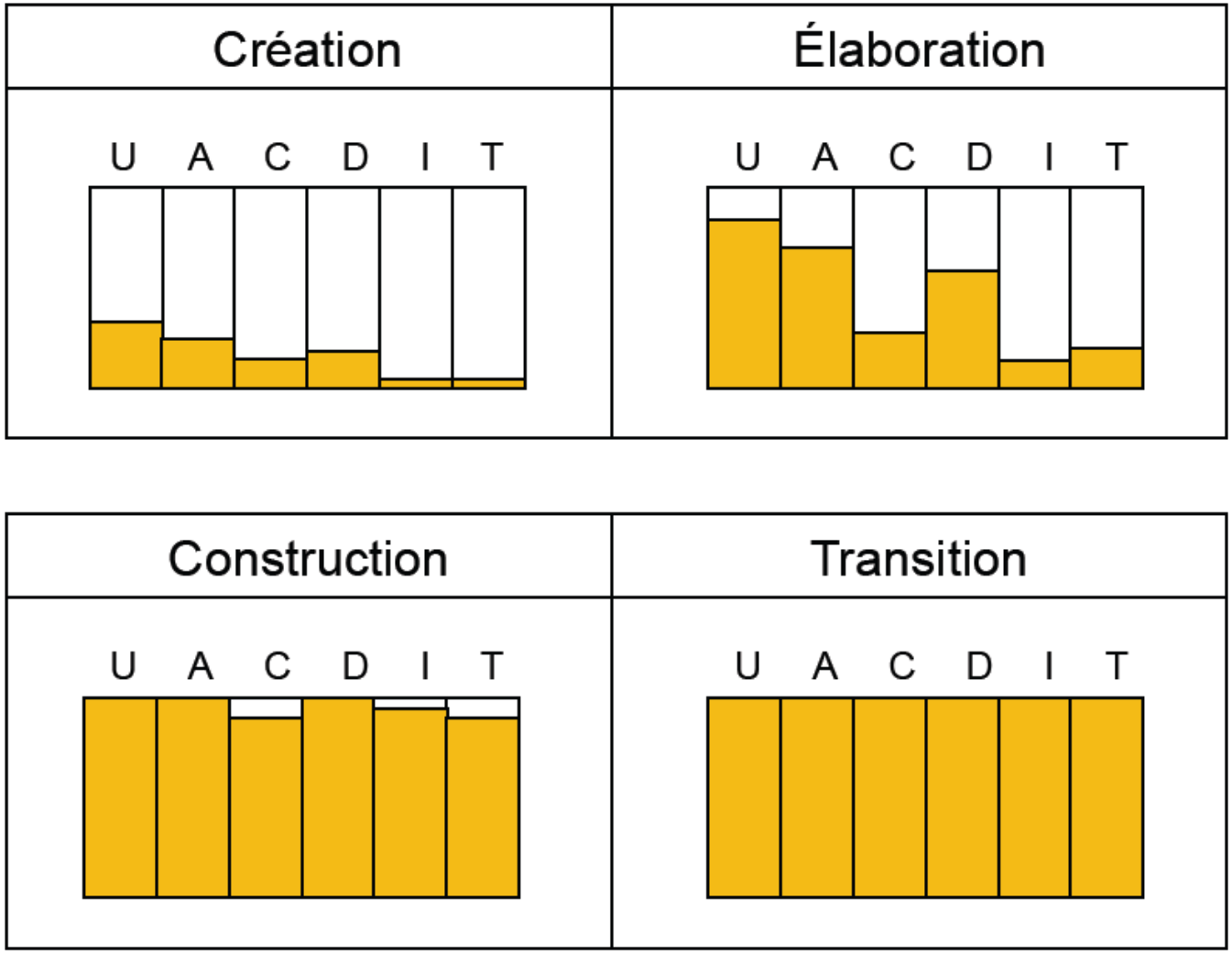
Activités principales de transition

- **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

Mini-conclusion

- UP
  - Est gros
  - Mais très strucurant
- Il décrit un ensemble de processus applicables
- Mais il faut s'adapter aux besoins du projet
- Exemples d'application (à venir)
  - 2TUP
  - UP « agiles »
- Et on peut faire de l'agile sans faire de l'UP…

Applications du processus unifié

Université Nice SOPHIA ANTIPOLIS

2TUP : Two Tracks Unified Process

- **Processus proposé par Valtech (consulting)**
  - Ref. : UML2 en action
- **Objectif**
  - prendre en compte les contraintes de changement continuel imposées aux systèmes d'information des organisations
- **Création d'un nouveau SI**
  - Deux grandes sortes de risques
    - imprécision fonctionnelle : inadéquation aux besoins
    - incapacité à intégrer les technologies : inadaptation technique
- **Evolution du SI**
  - Deux grandes sortes d'évolutions
    - évolution fonctionnelle
    - évolution technique

2TUP : principes

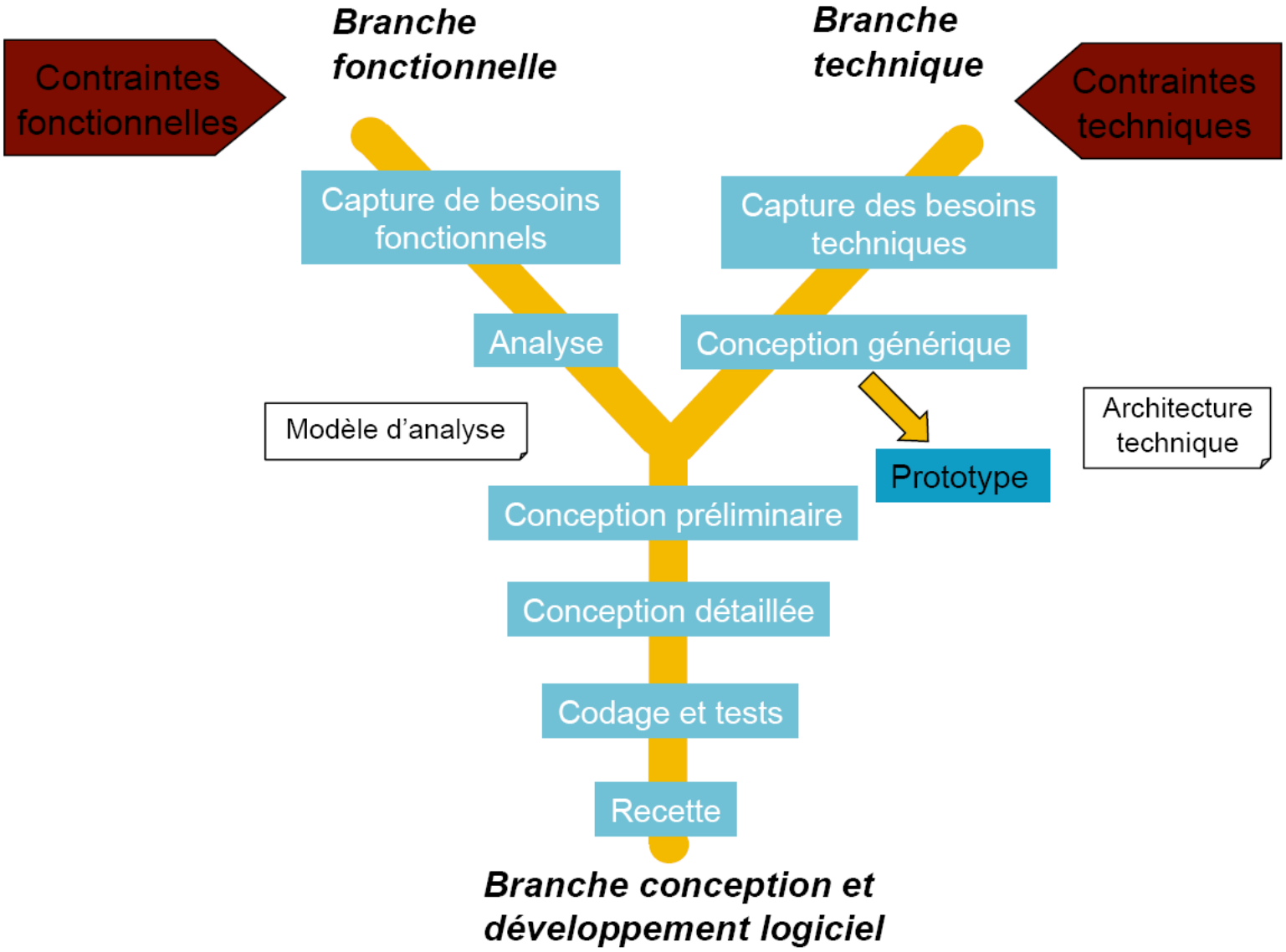
Principe général

- toute évolution imposée au système d'information peut se décomposer et se traiter parallèlement, suivant un axe fonctionnel et un axe technique

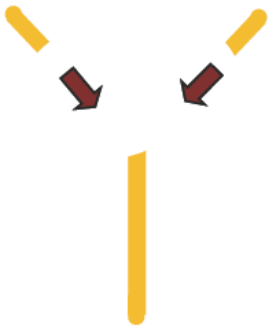
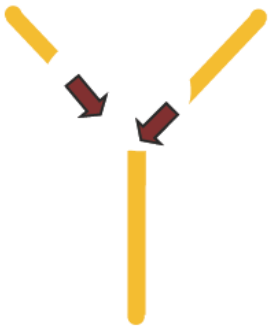
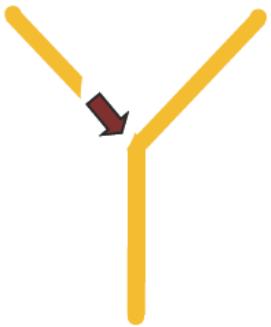
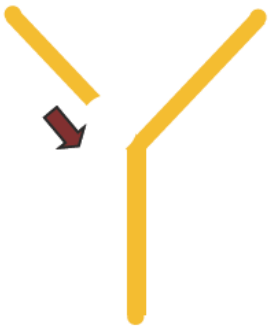
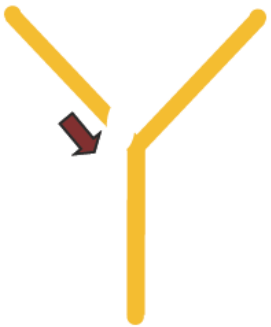
2TUP

- processus unifié (itératif, centré sur l'architecture et piloté par les CU)
- deux branches
  - besoins techniques : réalisation d'une architecture technique
  - besoins fonctionnels : modèle fonctionnel
- réalisation du système
  - fusionner les résultats des deux branches du processus
  - **C'est la partie la plus délicate !**

2TUP : schéma général



2TUP : itérations

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

Processus unifié « agile »

- **Proposé par Craig Larman**
  - *Ref : UML 2 et les Design Patterns* (3e édition)
- **Constat**
  - UP trop lourd, trop complexe, trop généraliste, trop d'artefacts, doit être adapté à chaque projet
  - Application d'un vieux modèle vertical, qui tend trop facilement vers la cascade
- **Principes de bonnes pratiques**
  - Itérations courtes (3 semaines max)
  - Mode organisationnel léger
    - petit ensemble d'activités et d'artefacts
  - Fusion analyse / conception
  - Utilisation de UML pour comprendre et concevoir plus que pour générer du code
  - Planification adaptative
  - Considère que UP est agile naturellement dans sa conception (et pour ses concepteurs), mais ne l'est pas dans ses applications

Méthodes « Agile »

Université Nice SOPHIA ANTIPOLIS

Plan

- Principes
- eXtreme Programming
- Scrum
- Conclusions

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

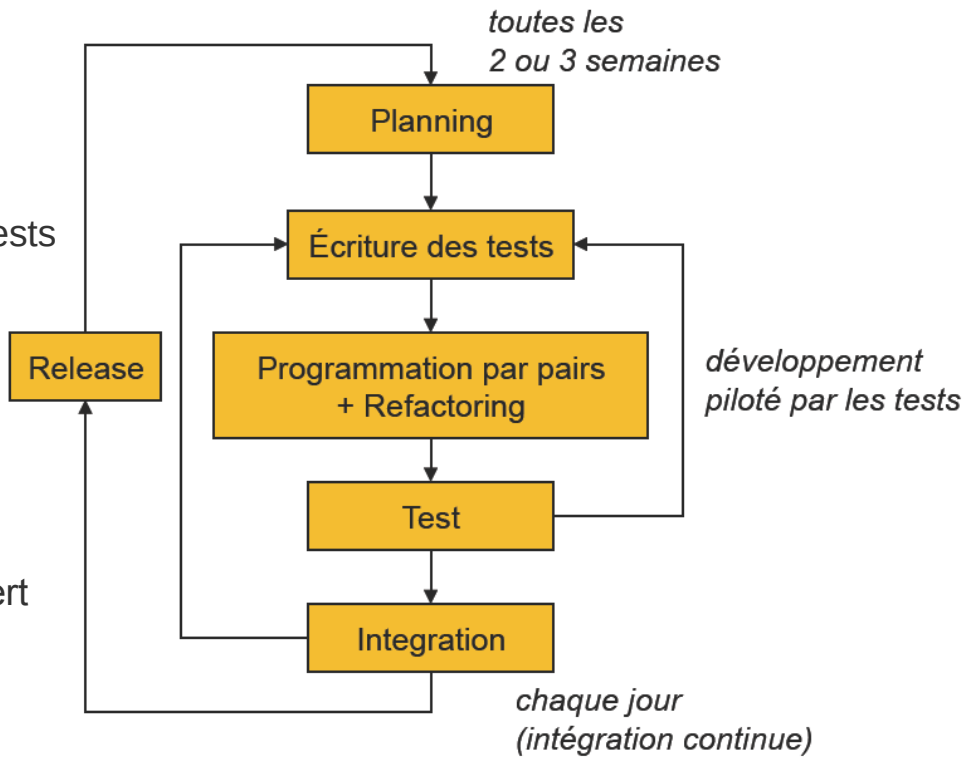
http://www.extremeprogramming.org

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

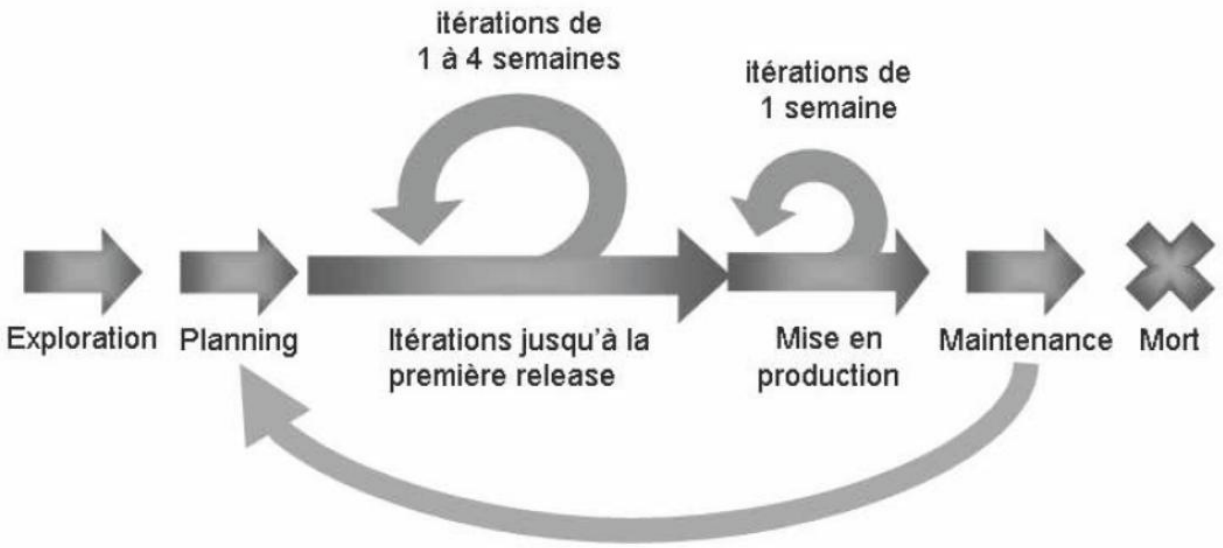
- **Client**
  - écrit les histoires et les tests fonctionnels
- **Testeur**
  - aide le client à écrire les tests, prépare les tests automatiques
- **Programmeur**
  - écrit les tests et puis code
- **Coach**
  - aide l'équipe par rapport au processus, expert méthode
- **Tracker**
  - suit les développement, 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

Scrum

- **Scrum : mêlée**
- **1986 -> 2001 ->…**
- **Phases**
  - Initiation / démarrage
    - Planning
      - définir le système : product Backlog = liste de fonctionnalités, ordonnées par ordre de priorité et d'effort
    - Architecture
      - conception de haut-niveau
  - Développement
    - Cycles itératifs (sprints) : 30j
      - amélioration du prototype
  - Clôture
    - Gestion de la fin du projet : livraison…

Scrum : principes

Isolement de l'équipe de développement

- l'équipe est isolée de toute influence extérieure qui pourrait lui nuire. Seules l'information et les tâches reliées au projet lui parviennent : pas d'évolution des besoins dans chaque sprint.

Développement progressif

- afin de forcer l'équipe à progresser, elle doit livrer une solution tous les 30 jours. Durant cette période de développement l'équipe se doit de livrer une série de fonctionnalités qui devront être opérationnelles à la fin des 30 jours.

Pouvoir à l'équipe

- l'équipe reçoit les pleins pouvoirs pour réaliser les fonctionnalités. C'est elle qui détient la responsabilité de décider comment atteindre ses objectifs. Sa seule contrainte est de livrer une solution qui convienne au client dans un délai de 30 jours.

Contrôle du travail

- le travail est contrôlé quotidiennement pour savoir si tout va bien pour les membres de l'équipe et à la fin des 30 jours de développement pour savoir si la solution répond au besoin du client.

Scrum : rôles et pratiques

- **Scrum Master**
  - expert de l'application de Scrum
- **Product owner**
  - responsable officiel du projet
- **Scrum Team**
  - équipe projet.
- **Customer**
  - participe aux réunions liées aux fonctionnalités
- **Management**
  - prend les décisions
- **Product Backlog**
  - état courant des tâches à accomplir
- **Effort Estimation**
  - permanente, sur les entrées du backlog
- **Sprint**
  - itération de 30 jours
- **Sprint Planning Meeting**
  - réunion de décision des objectifs du prochain sprint et de la manière de les implémenter
- **Sprint Backlog**
  - Product Backlog limité au sprint en cours
- **Daily Scrum meeting**
  - ce qui a été fait, ce qui reste à faire, les problèmes
- **Sprint Review Meeting**
  - présentation des résultats du sprint

Conclusions

Conclusions

- Il faut trouver le bon processus
- Il faut l'adapter
- Et qu'est-ce qu'on gagne ?
  - 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.