

Evolution des projets (TD)

Avancement des projets

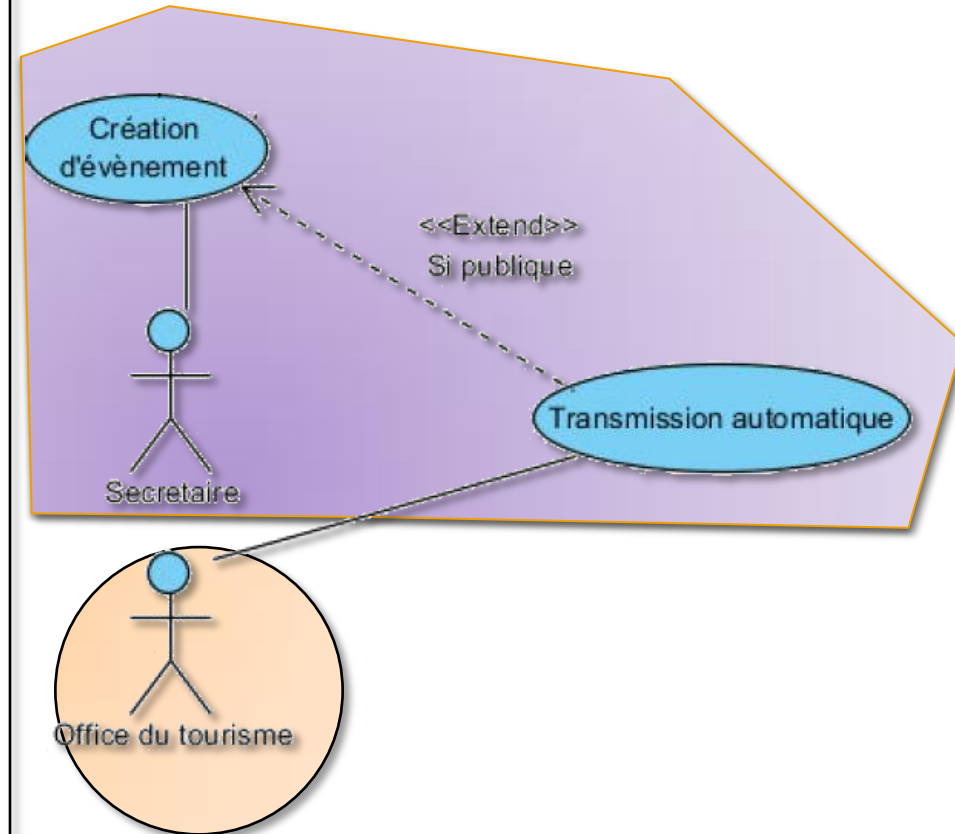
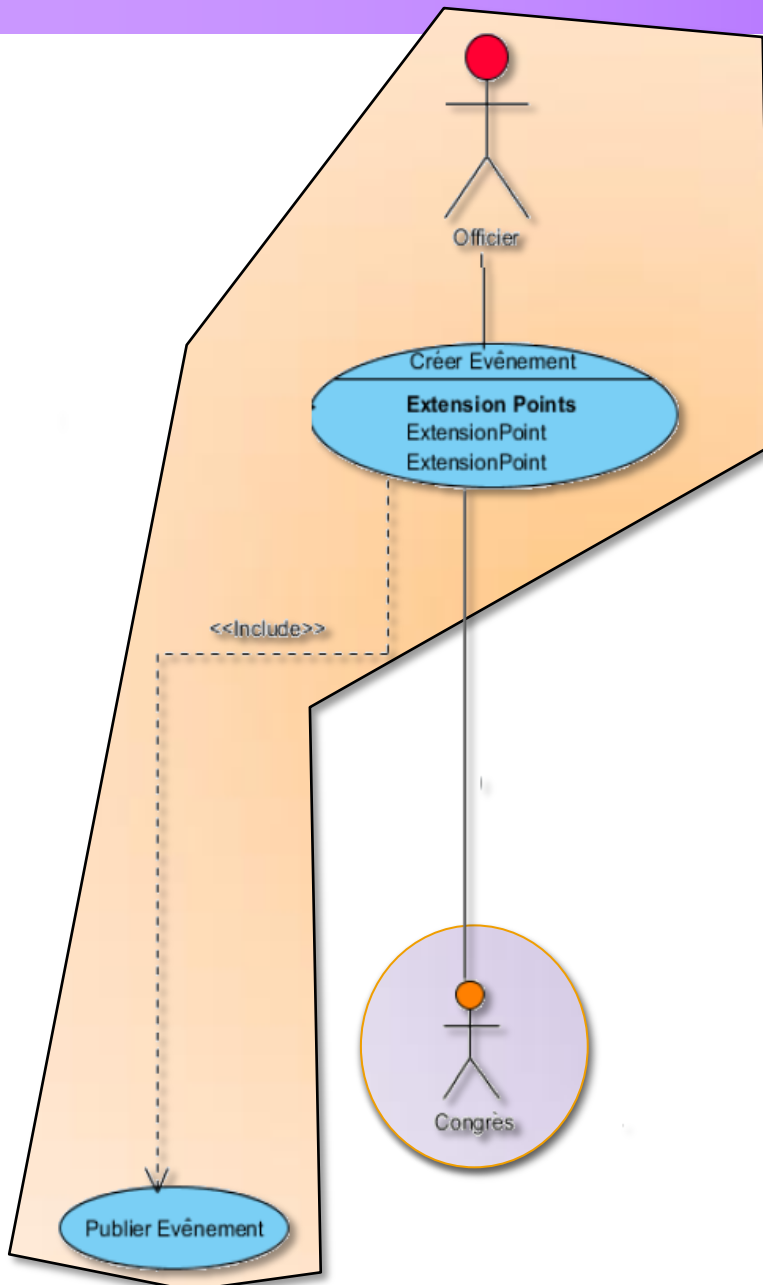
- ❑ **Ce qui devrait être finit**
 - Use Case (vue globale)
 - Scénario
 - Détail sur le noyau (partie centrale du S.I.)
 - Idée des Maquettes (IHM) pour le noyau

- ❑ **Dans les 3 semaines à venir (tâches parallèles)**
 - Diagramme de Classes pour le noyau
 - Diagramme de séquences pour le noyau
 - Affinement des maquettes
 - Discussion avec au moins un autre projet
 - Élaboration rapport

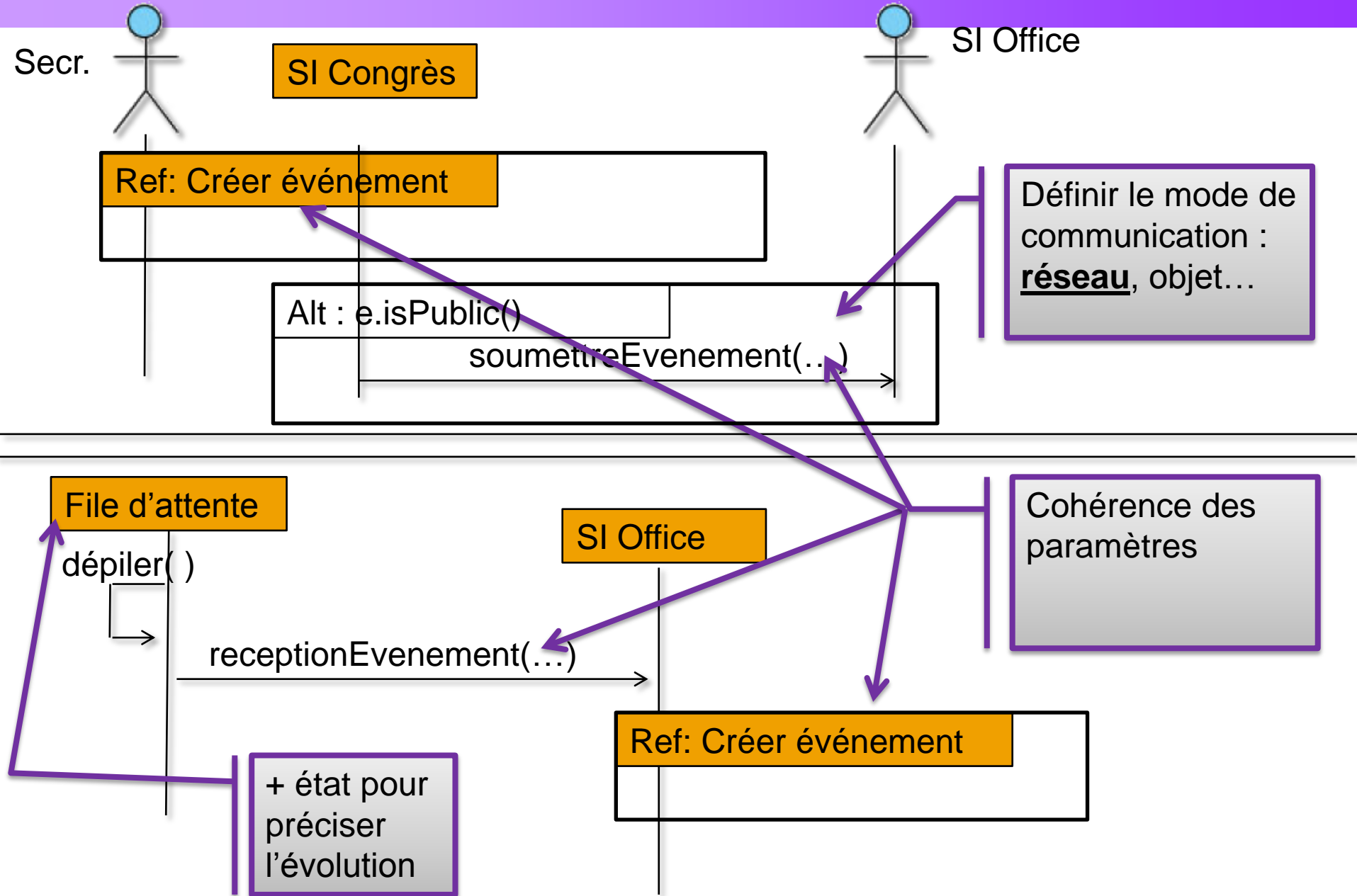
Collaboration entre S.I.

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

Correspondance de Use Case



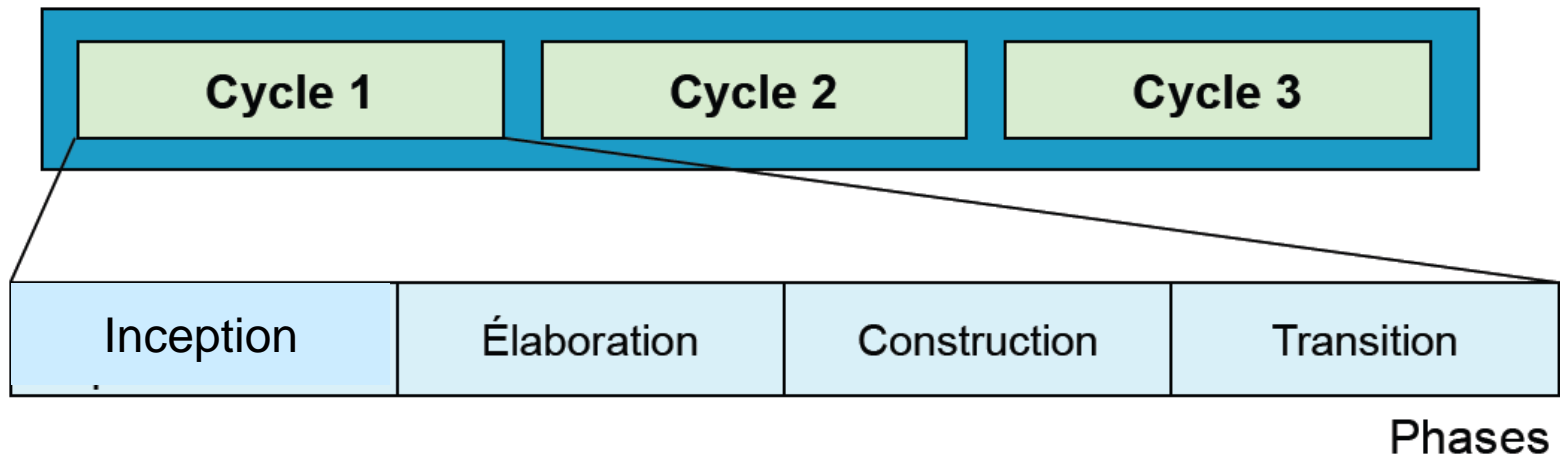
Correspondance de Séquences



Rappel sur UDSP

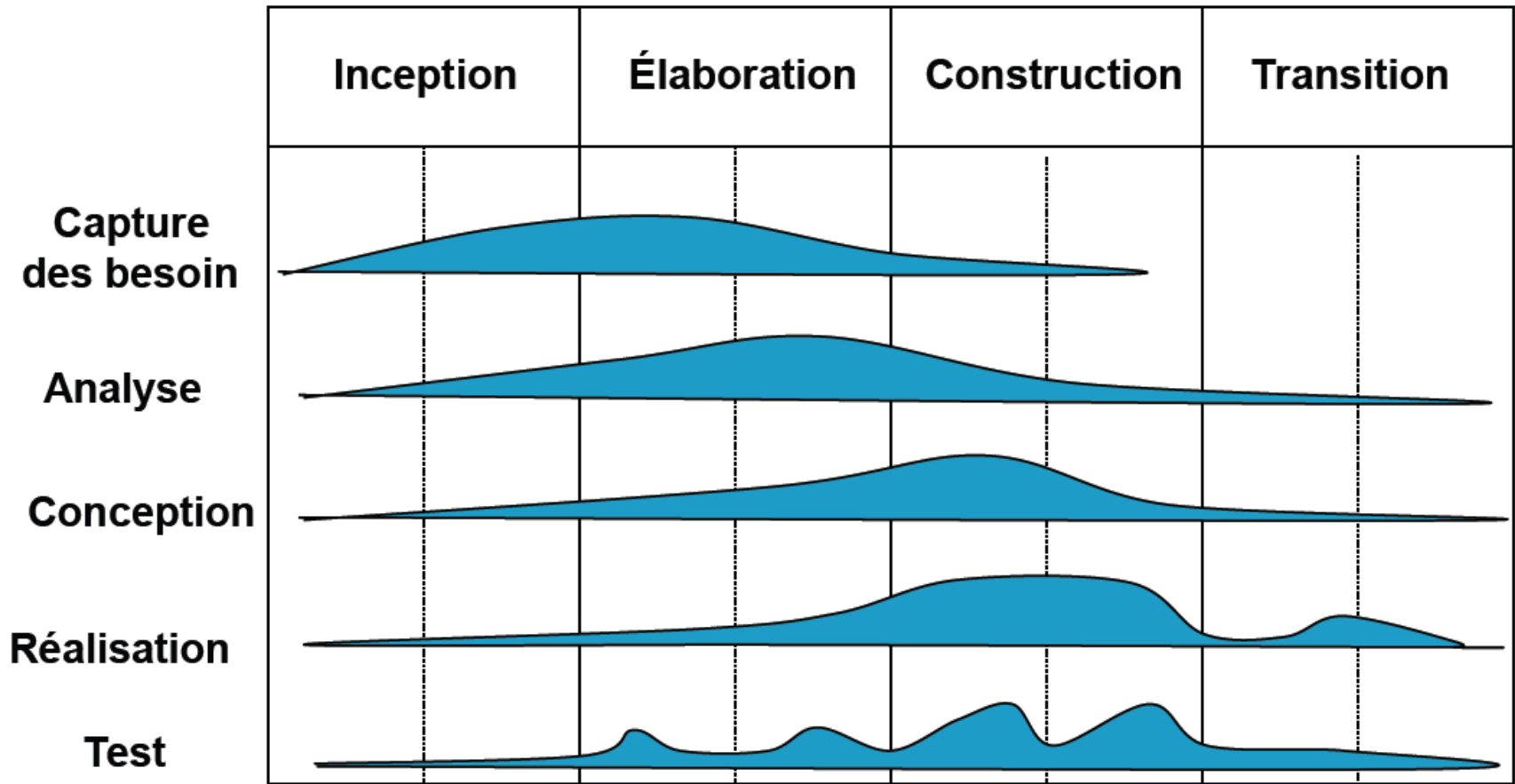
USDP : Principes

- ❑ 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

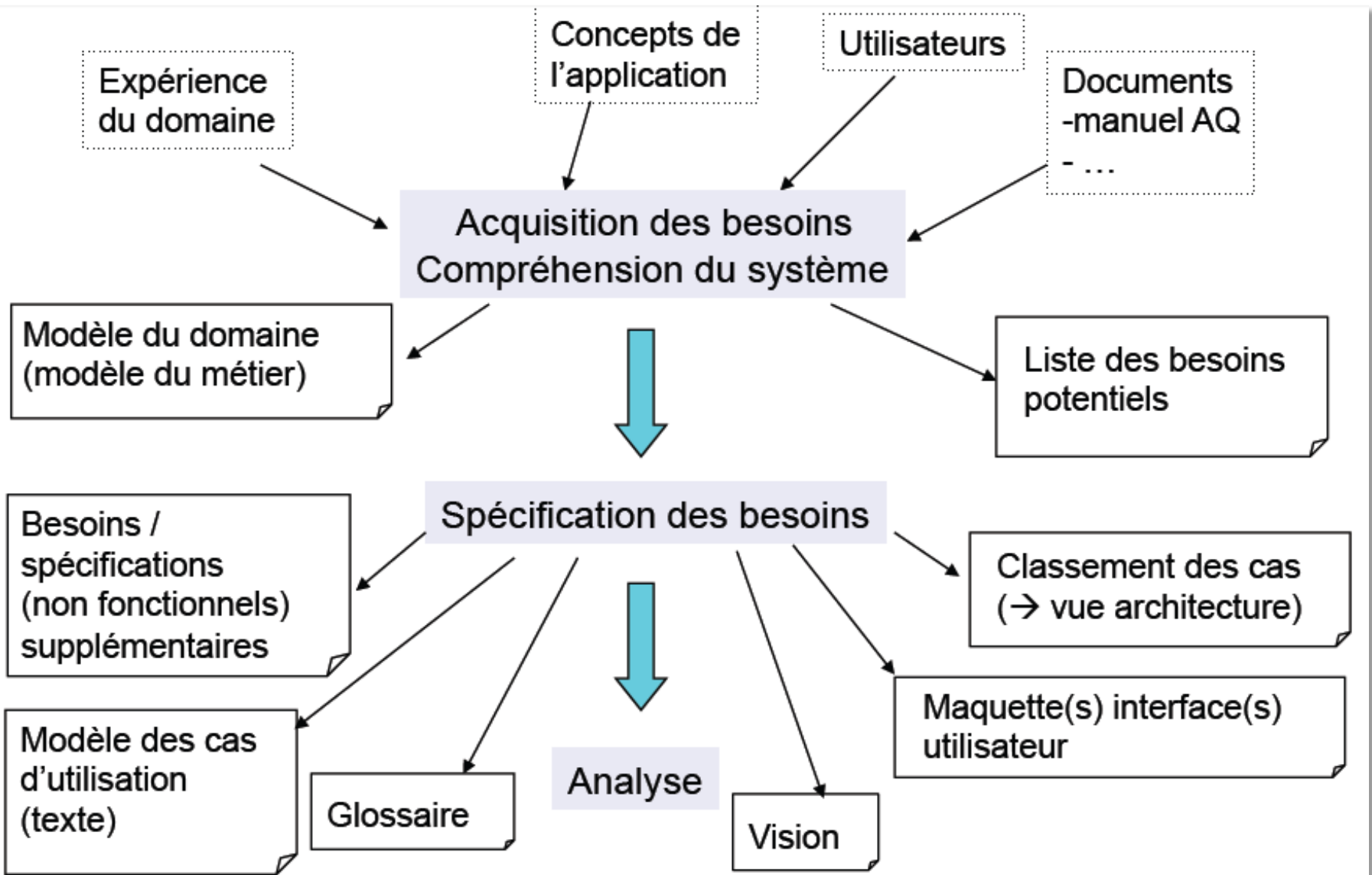


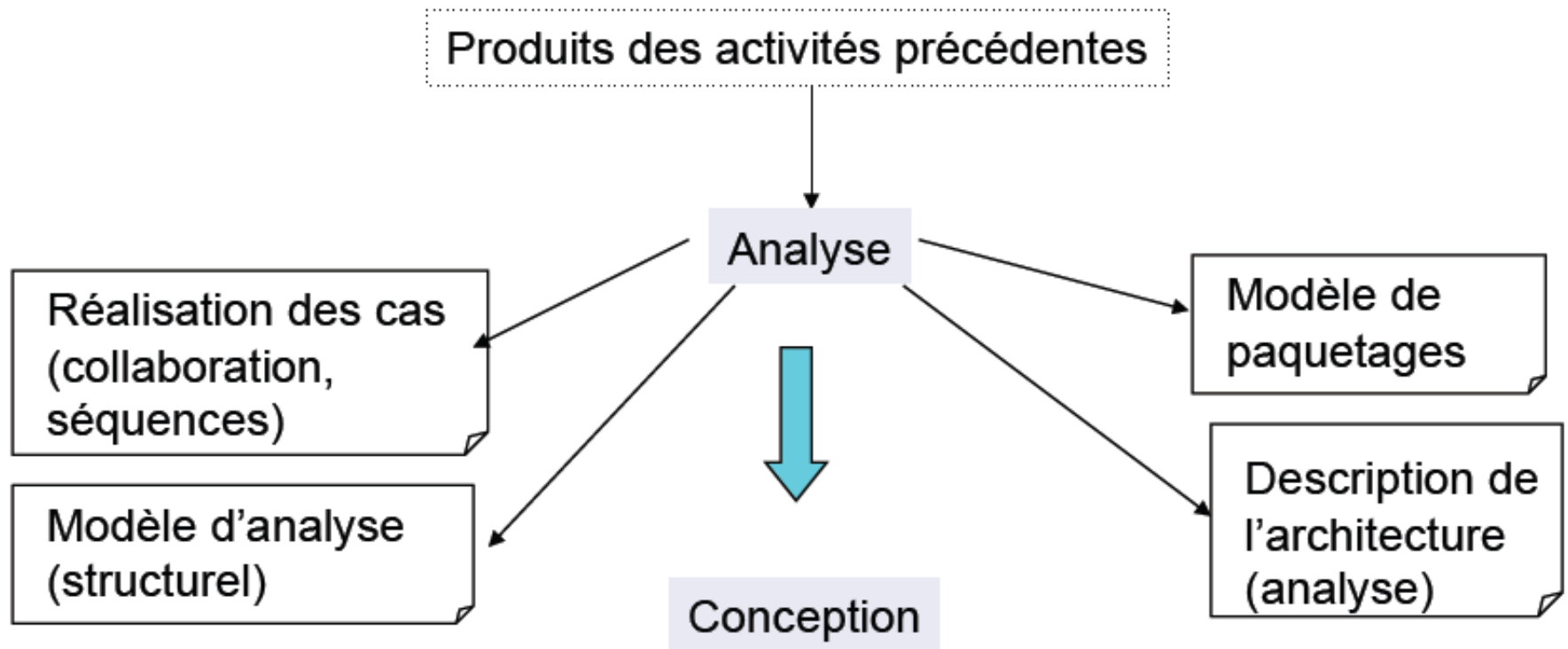
Effort lors des phases

- Chaque phase spécifie les activités à effectuer

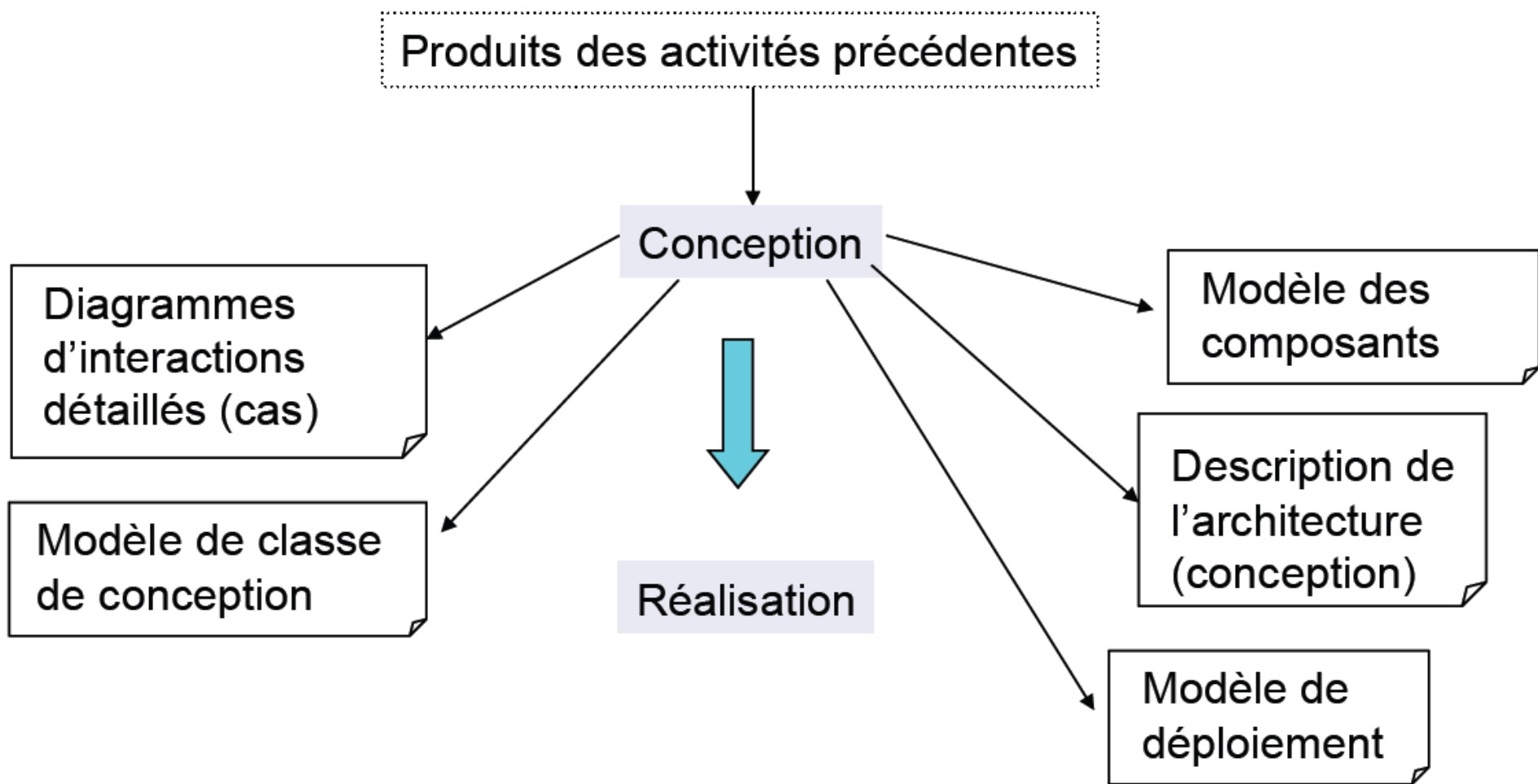


Productions

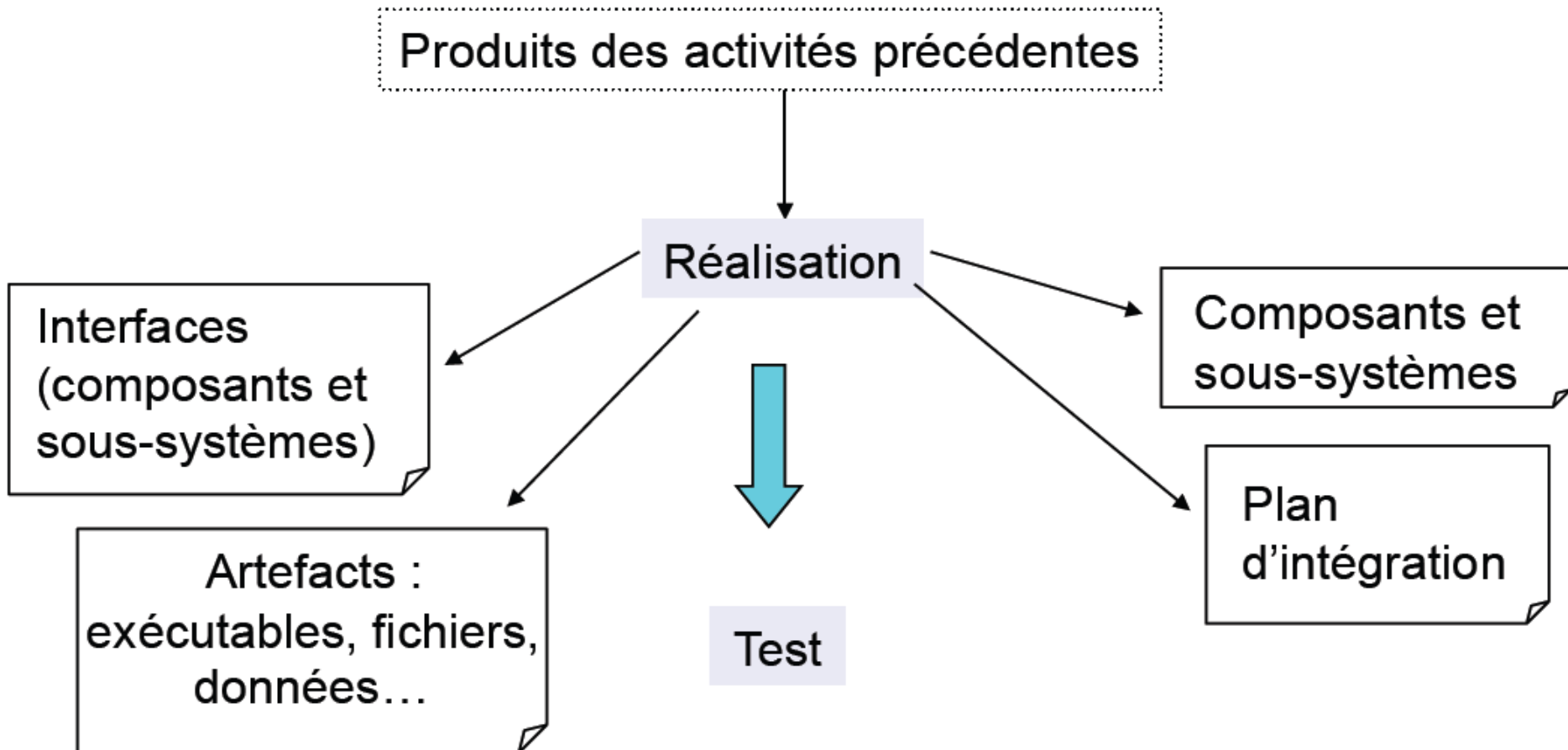




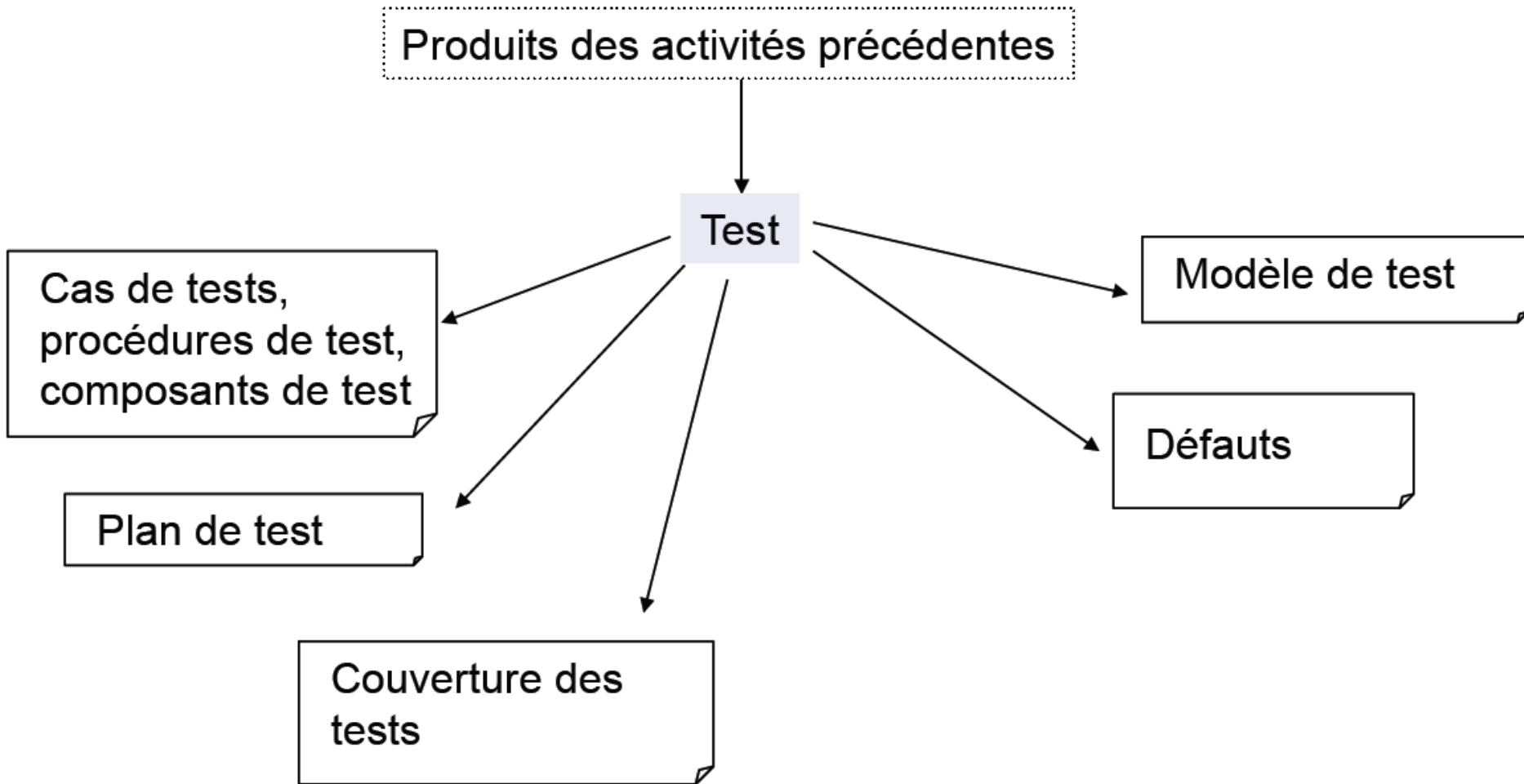
Productions



Productions



Productions



Applications du processus unifié

2TUP : Two Tracks Unified Process

❑ Processus proposé par Valtech (consulting)

- Ref. : UML2 en action

❑ Objectif

- prendre en compte les contraintes de changement continu 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

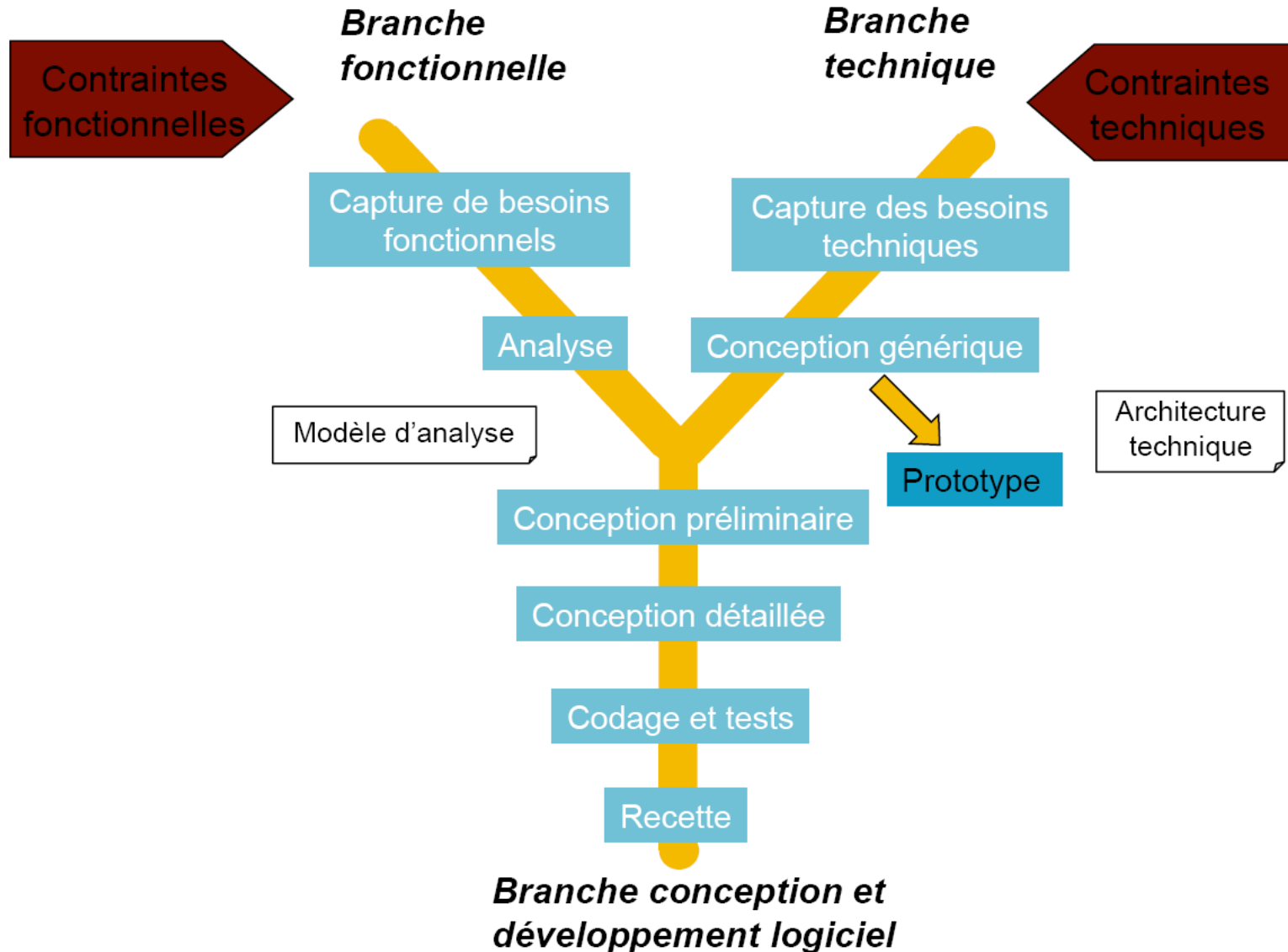
□ Principe général

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

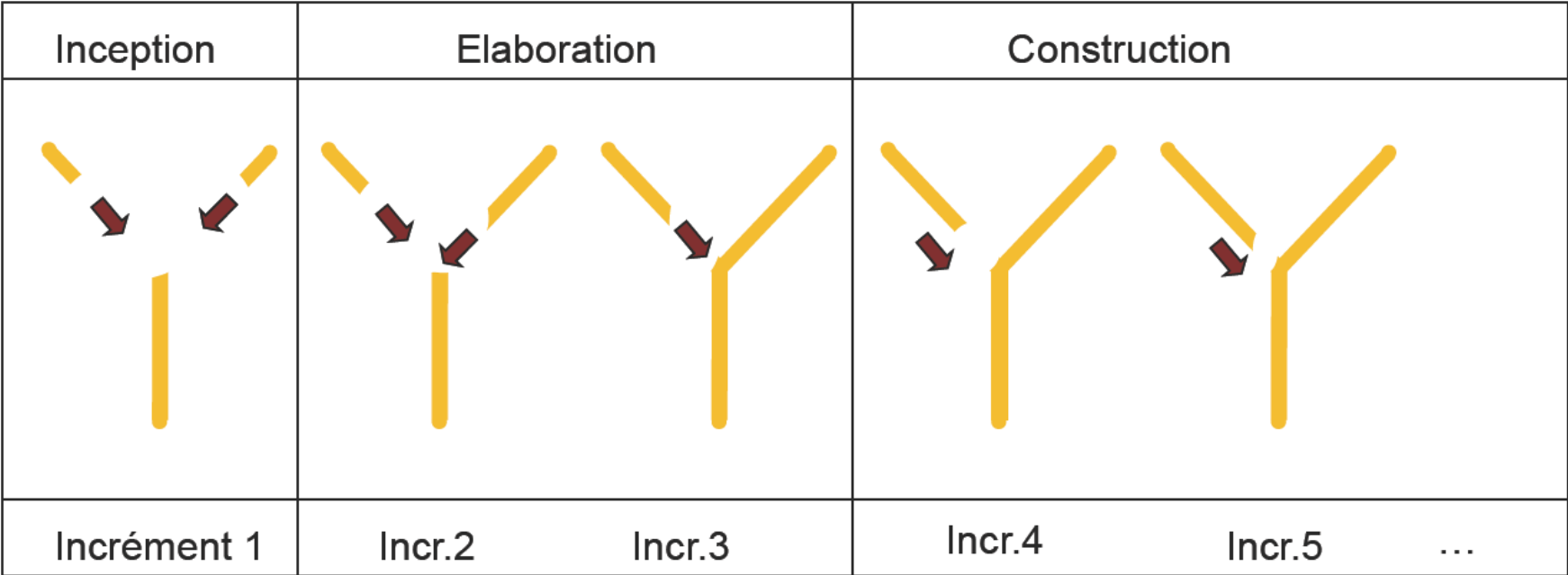
□ 2TUP

- 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



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 »

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

❑ Méthodes adaptatives (vs. prédictives)

- itérations courtes
- lien fort avec le client
- fixer les délais 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 customer 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

❑ Caractéristiques principales

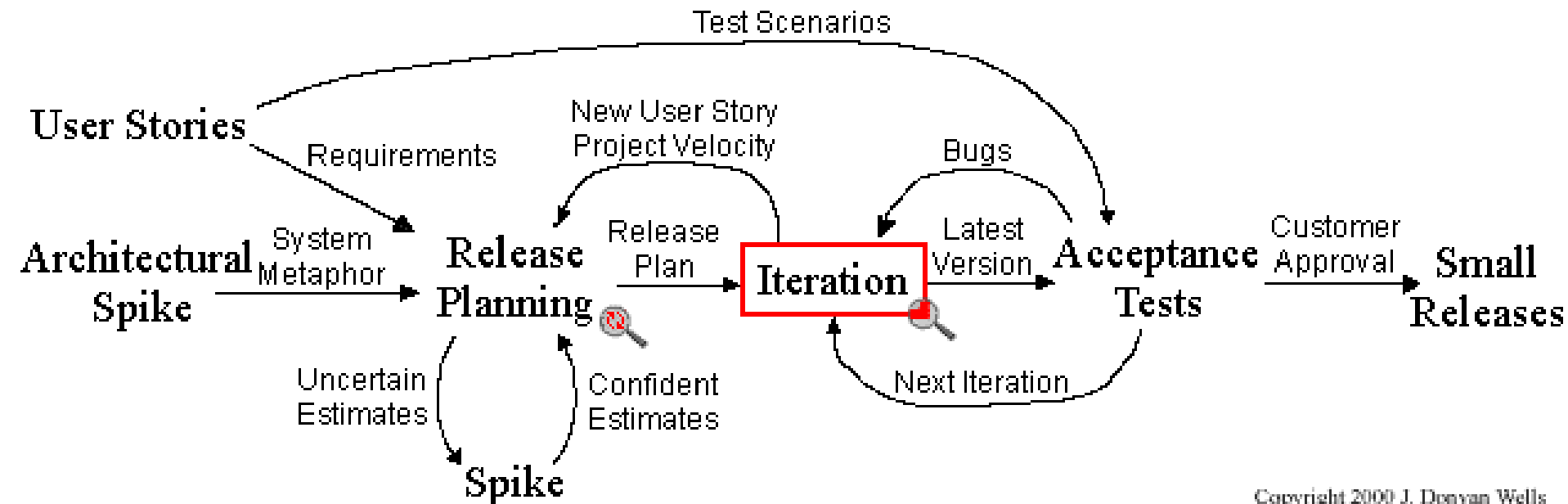
- Le client (maîtrise d'ouvrage) pilote lui-même le projet, et ce de très près grâce à des cycles itératifs extrêmement courts (1 ou 2 semaines).
- L'équipe autour du projet livre très tôt dans le projet une première version du logiciel, et les livraisons de nouvelles versions s'enchaînent ensuite à 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



Extreme Programming Project



Copyright 2000 J. Donovan Wells

“A spike solution is a very simple program to explore potential solutions. Build the spike to only address 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

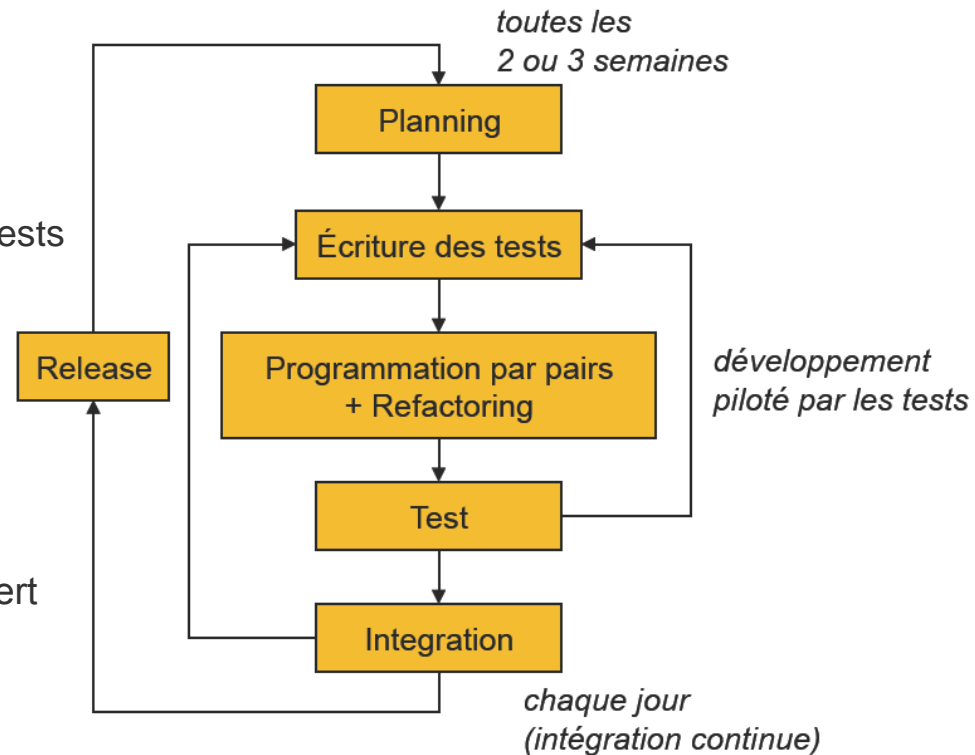
- suit les développement, vérifie que l'équipe ne perd pas la bonne direction

☐ Manager

- Responsable infrastructure et soucis extérieurs

☐ Consultant

- fournit les connaissances spécialisées au besoin

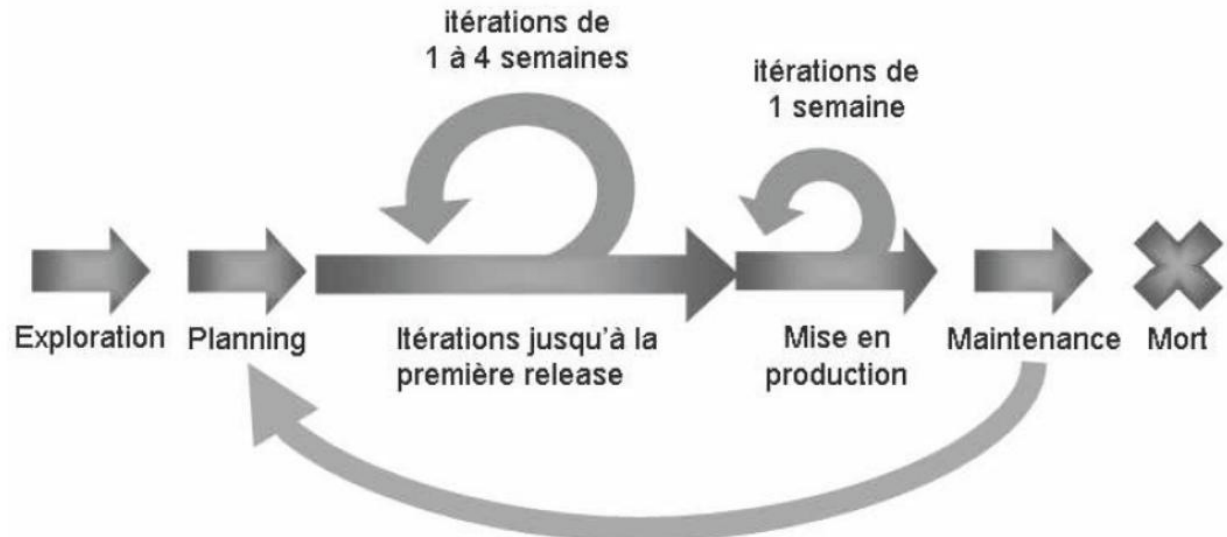


□ Planification

- regroupement des intervenants pour planifier l'itération
- les développeurs évaluent les risques techniques et les efforts prévisibles liés à chaque fonctionnalité (user story = sortes de scénarios abrégés)
- les clients estiment la valeur (l'urgence) des fonctionnalités, et décident du contenu de la prochaine itération

□ Temps court entre les releases

- au début : le plus petit ensemble de fonctionnalités utiles
- puis : sorties régulières de prototypes avec fonctionnalités ajoutées



❑ Métaphore

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

❑ 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

❑ 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

❑ 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

□ Avantages

- Concept intégré et simples
- Pas trop de management
 - ◆ Pas de procédures complexes, ni de doc à maintenir
 - ◆ communication directe
 - ◆ programmation par paires
- Gestion continue 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 : 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 liées 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 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.