Université Nice Sophia Antipolis

Analyse des besoins et cahier des charges

- Terminologie
- La faisabilité
- L ’analyse des besoins
- Le cahier des charges

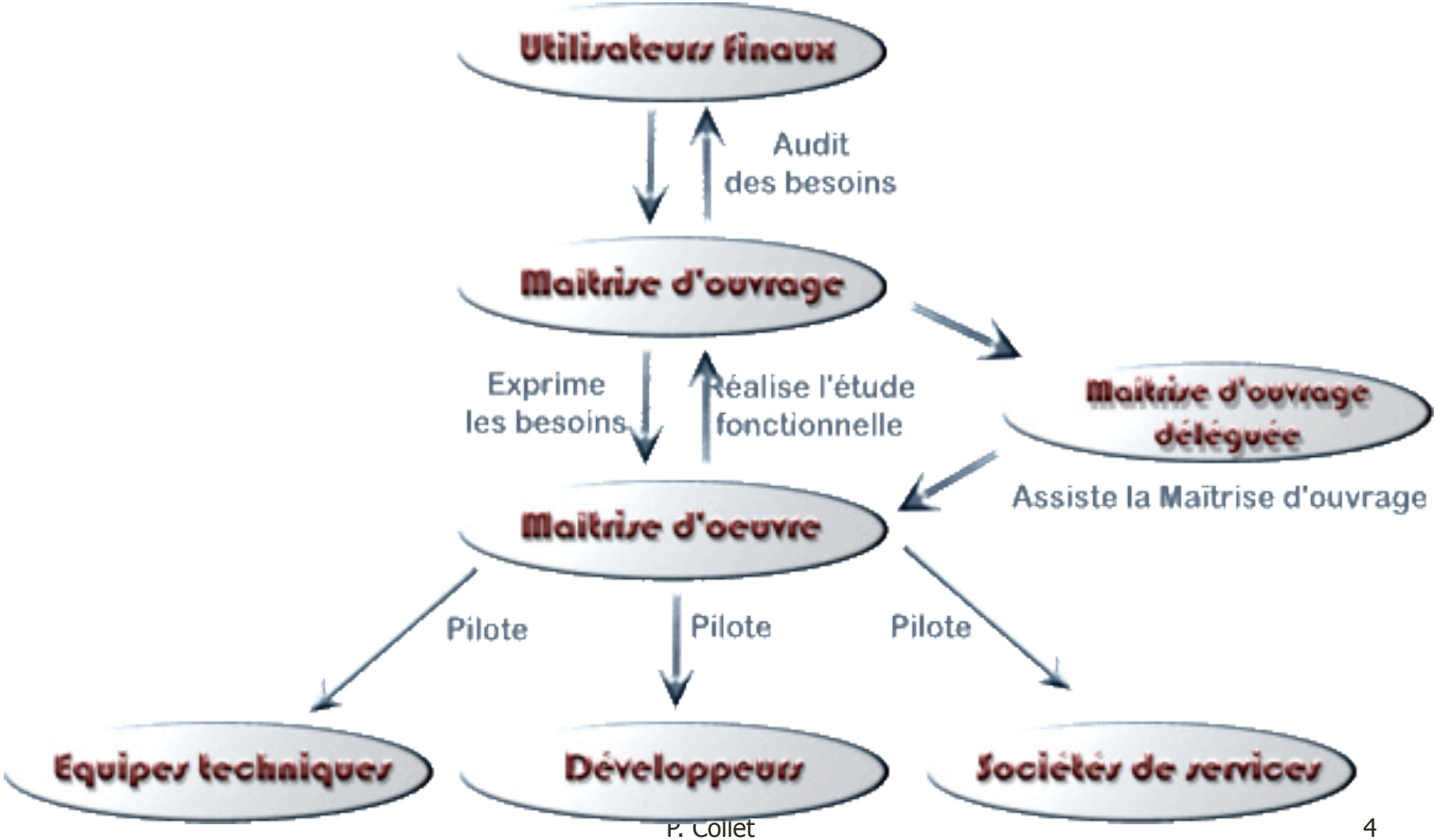
Système informatique

- "Un ensemble d'éléments qui sont organisés pour accomplir un but prédéfini par un traitement de l'information"
- utilise des :
  - Logiciels
  - Matériels (informatiques)
  - Personnes
  - Bases de données (ensemble *organisée* de données)
  - Documentation
  - Procédures (étapes qui définissent comment utiliser les éléments du système)

Développement d'un système

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

Différence dans les *maîtrises*



Cahier des charges / Description of Work

- Résumé exécutif
  - une demi page pour aller à l 'essentiel avec objectifs attendus
- 1. Description du projet
  - Contexte de travail
    - Environnement, positionnement
  - Motivations
    - Bien fondé du projet, exemples
  - Défis
    - Points difficiles (défi global, puis sous-points)
  - Objectifs
    - Objectifs qui seront évalués en fin de projet
  - Scénario(s)
    - 1 ou plusieurs scénarios expliquand comme les résultats du projet pourront être appliqués sur des cas concrets
  - Critères de succès
    - Comment évaluer le projet vis-à-vis des objectifs

SPECIALISTES

Cahier des charges / Description of Work

- 2. Etat de l'art
  - Description générale
    - Ecosystème du projet (technologies)
    - Outils utilisés usuellement pour traiter le prob
- 3. Méthodologie et planification
  - Stratégie générale
    - Modèle de cycle de vie (cascade, itération)
    - Phases de mise en œuvre et lien entre les phases
  - Découpage en lots
    - Lots du projet avec un numéro, titre, type de travail, nom du responsable…

SPECIALISTES

Cahier des charges / Description of Work

- Planification
  - Diagramme de Gantt du projet (cf. fin du cours)
- Livrables associés au projet
  - Liste des livrables, lot associé, nature (document, code, etc.)
- Jalons
  - Point de vérification du projet (typiquement livraison)
- Pilotage et suivi
  - Principes de pilotage (durée des itérations pour un pilotage agile, moyen et personnel pour une approche plus classique…)

SPECIALISTES

Cahier des charges / Description of Work

- 4. Description de la mise en œuvre du projet
  - Interdépendances des lots et tâches
  - Description des lots (objectif, contenu, livrable)
  - Résumé de l'effort

  => dans JIRA

SPECIALISTES

- 5. Participants
  - Liste des personnels

SPECIALISTES

- 6. Bibliographie, références, acronymes

Université Nice Sophia Antipolis

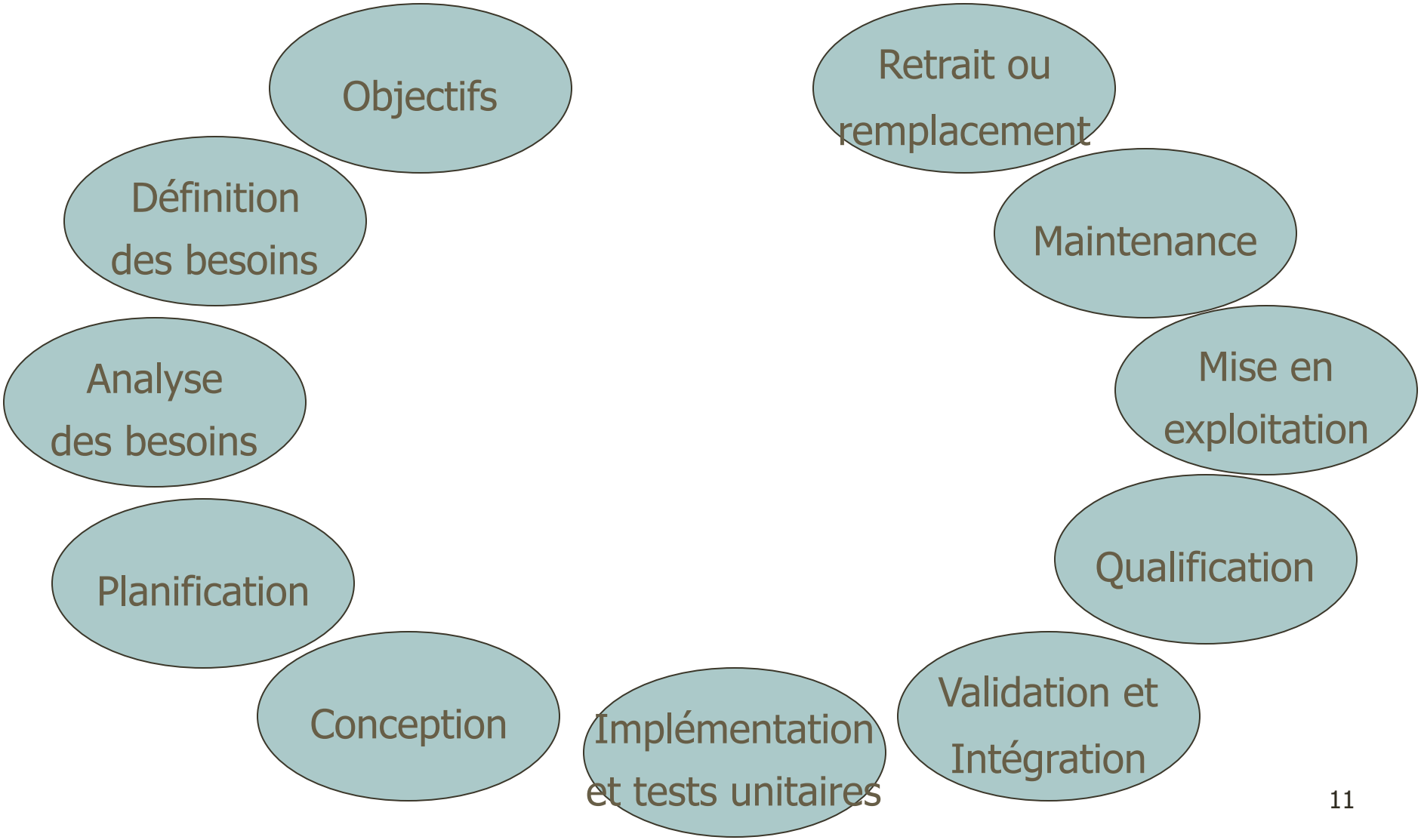
Cycle de vie du logiciel

- Les phases du cycle de vie
- Les modèles de développement

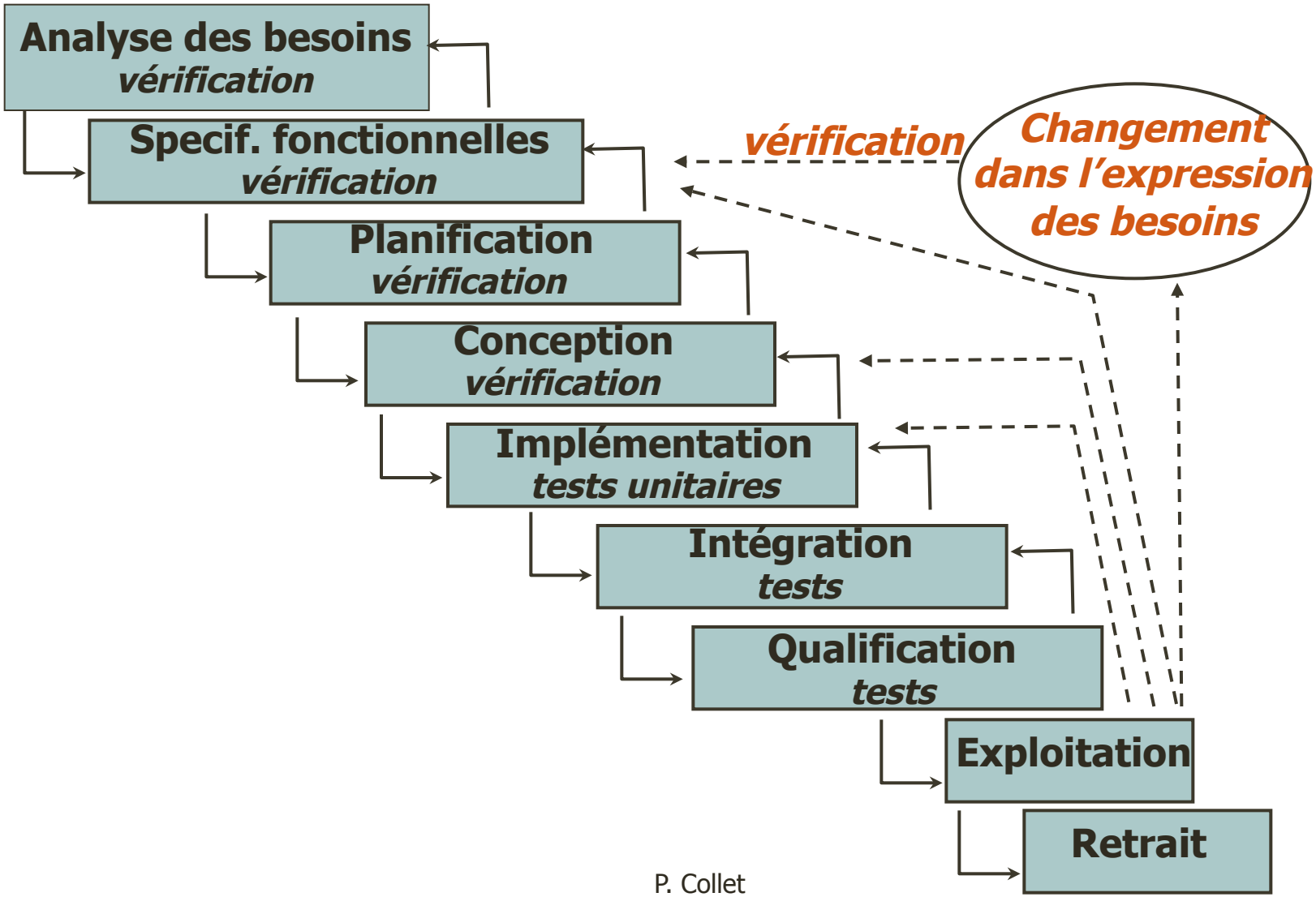
Notion de cycle de vie

- 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 ?

Les phases du cycle de vie



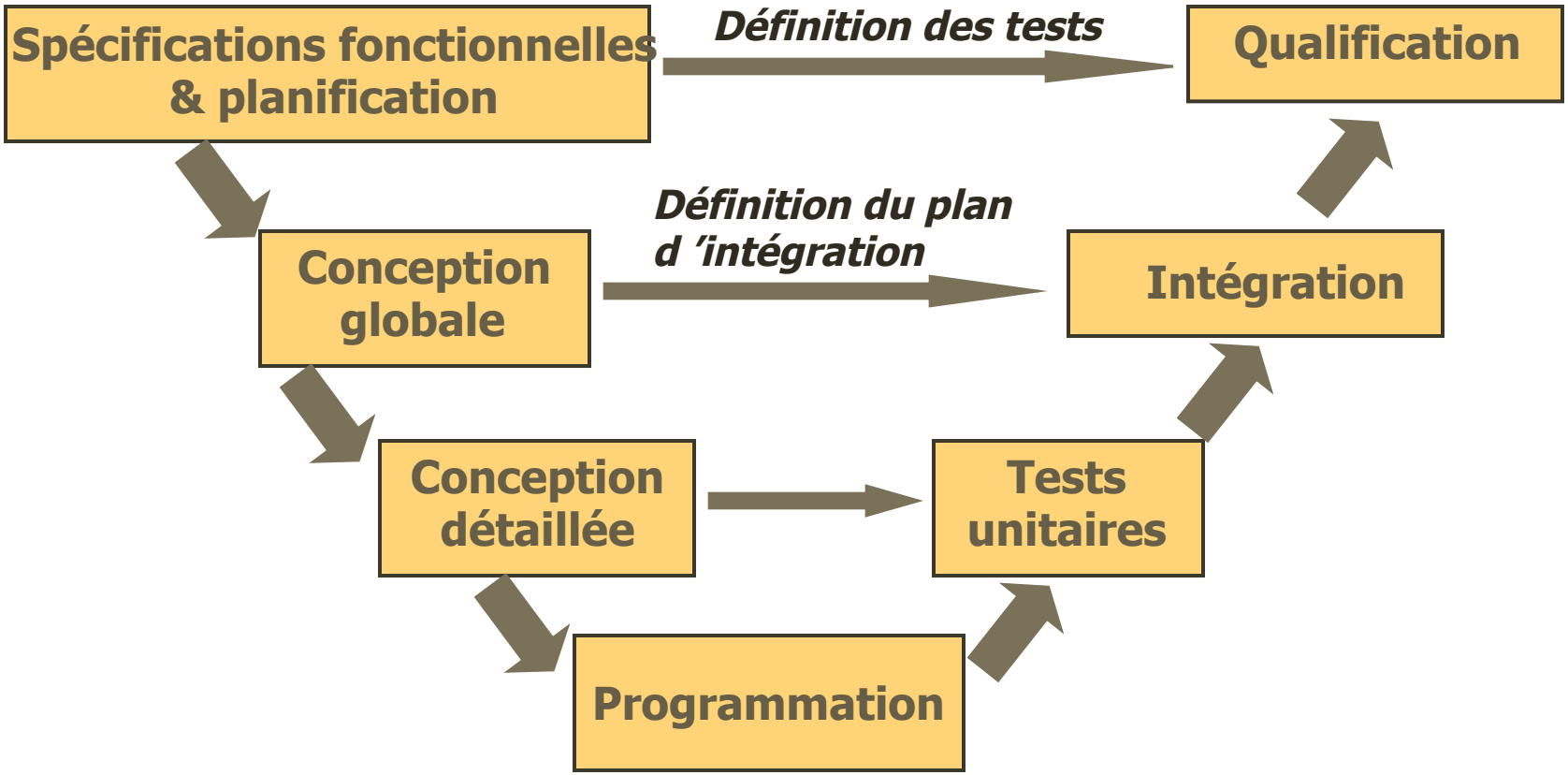
Modèle en cascade (1970)



Problèmes du modèle en cascade

- Les vrais projets suivent rarement un développement séquentiel
- Établir tous les besoins au début d'un projet est difficile
- Le produit apparaît tard
- Seulement applicable pour les projets qui sont bien compris et maîtrisés

Modèle en V

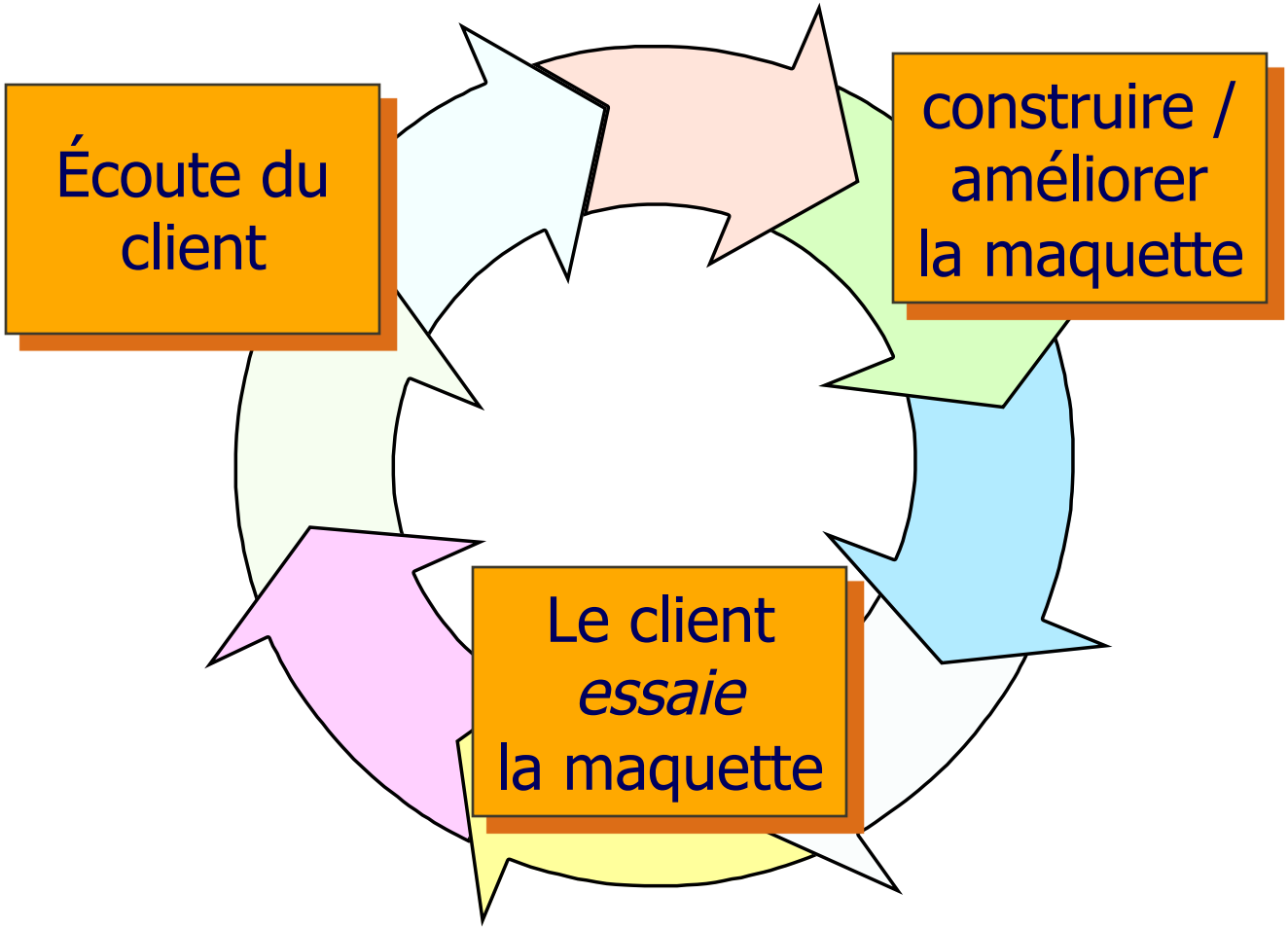


☞ *Gestion des configurations, de projet, plan assurance qualité*

Comparaison

- Le cycle en V
  - permet une meilleure anticipation
  - évite les retours en arrière
- Mais
  - le cadre de développement est rigide
  - la durée est souvent trop longue
  - le produit apparaît très tard

Prototypage



Prototypage, RAD

RAD : *Rapid Application Development*

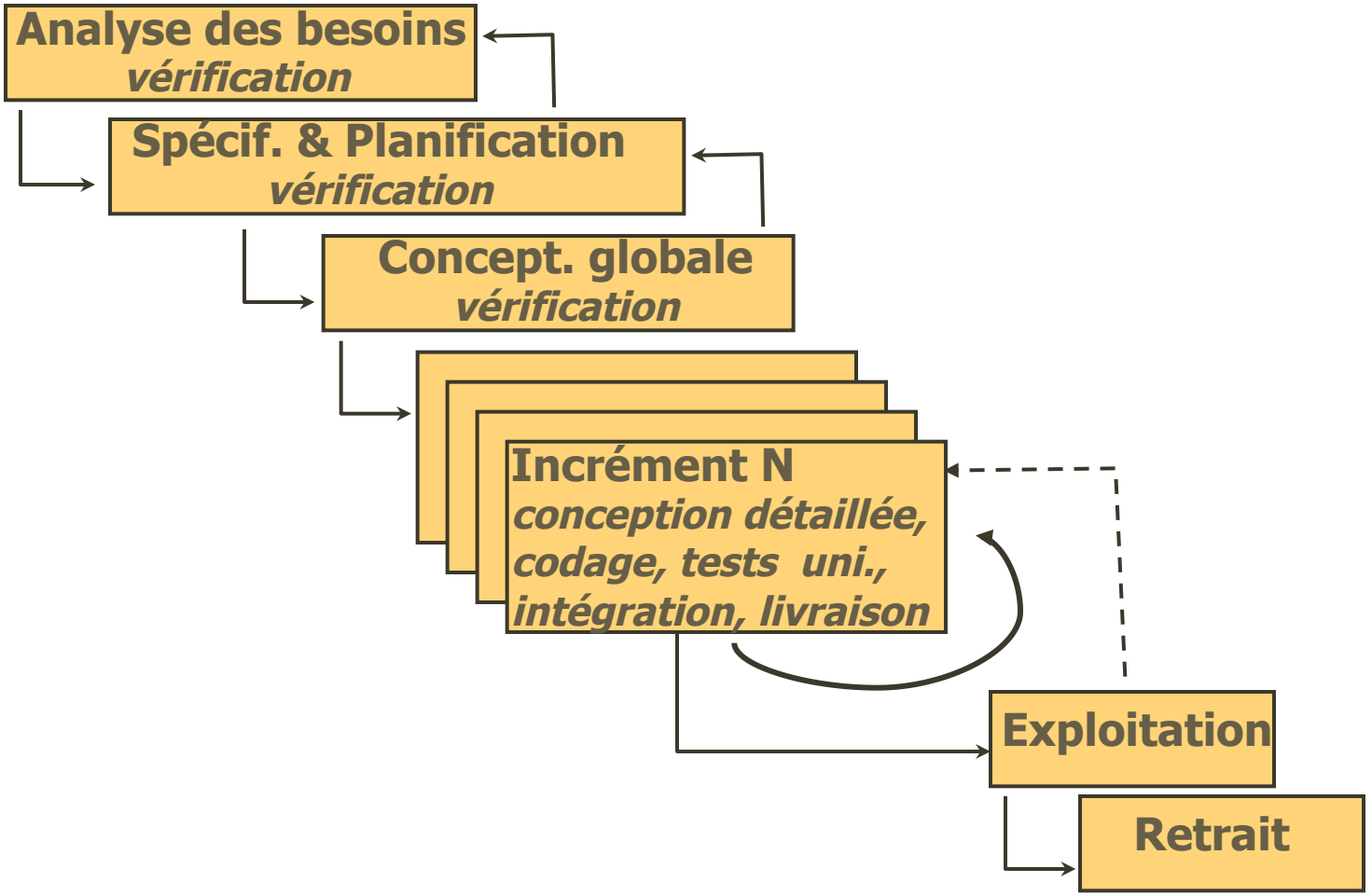
- Discuter et interagir avec l'utilisateur
- Vérifier l 'efficacité réelle d 'un algorithme
- Vérifier des choix spécifiques d 'IHM
- Souvent utilisé pour identifier les besoins
  - Prototype jetable (moins de risque ?)
- Souvent implémenté par des générateurs de code
  - Prototype évolutif

Prototypage, RAD (suite)

- Mais :
  - Les objectifs sont uniquement généraux
  - Prototyper n'est pas spécifier
  - Les décisions rapides sont rarement de bonnes décisions
  - Le prototype évolutif donne-t-il le produit demandé ?
  - Les générateurs de code produisent-ils du code assez efficace ?

☞ *Projets petits ou à courte durée de vie*

Modèle incrémental



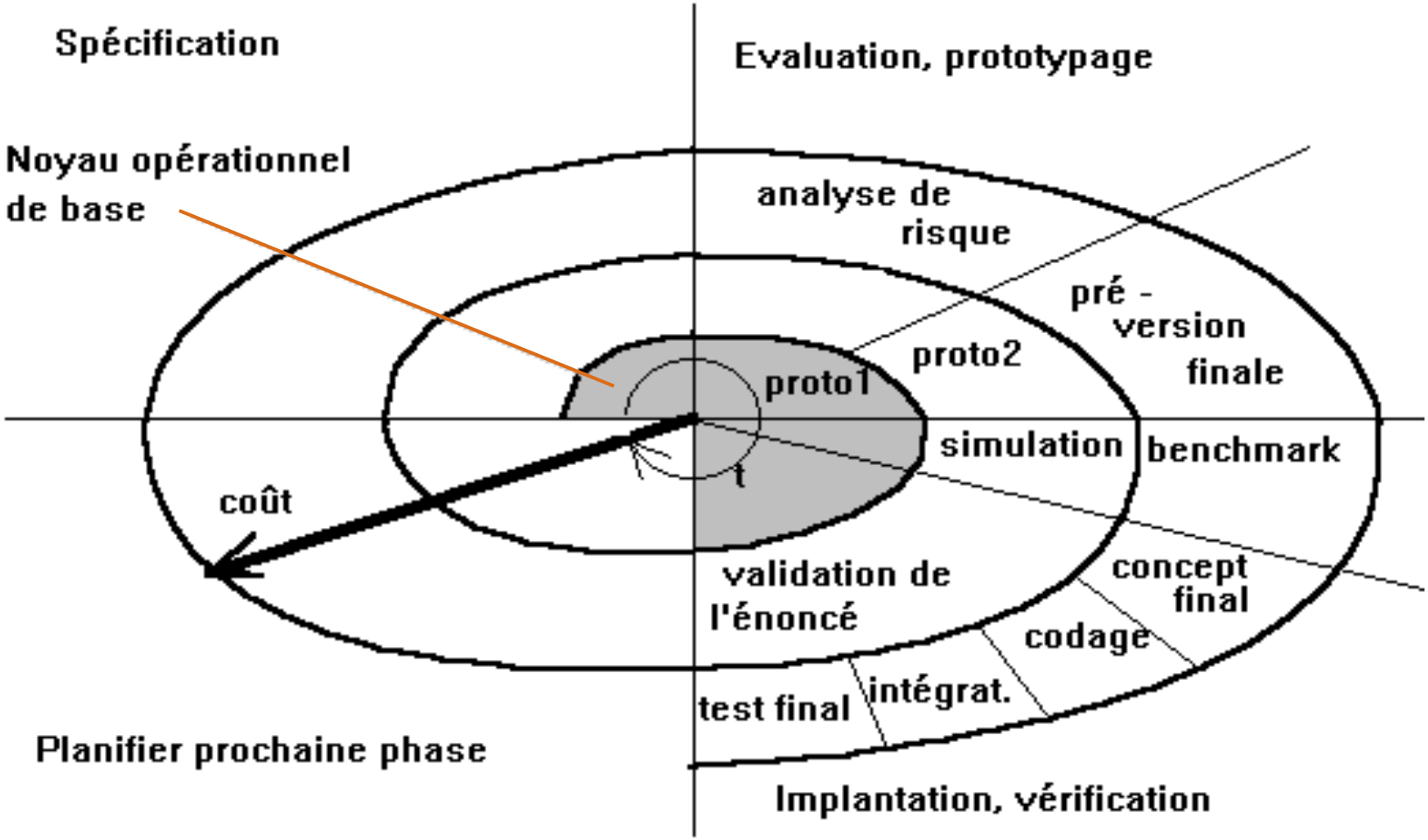
Le développement incrémental

- combine des éléments des modèles linéaires et du prototypage
- produit des incréments *livrables*
- se concentre sur un produit opérationnel (pas de prototype jetable)
- peut être utilisé quand il n'y a pas assez de ressources disponibles pour une livraison à temps

☞ *Le premier incrément est souvent le noyau*

☞ *Les incréments aident à gérer les risques techniques (matériel non disponible)*

Modèle en spirale (Boehm, 1988)



Modèle en spirale (suite)

- Spécification : communiquer avec le client
- Analyse de risque : évaluation des risques techniques et des risques de gestion
- Implémentation et vérification : construire, tester, installer et fournir un support utilisateur
- Validation: obtenir des *retours*
- Planification : définir les ressources, la répartition dans le temps

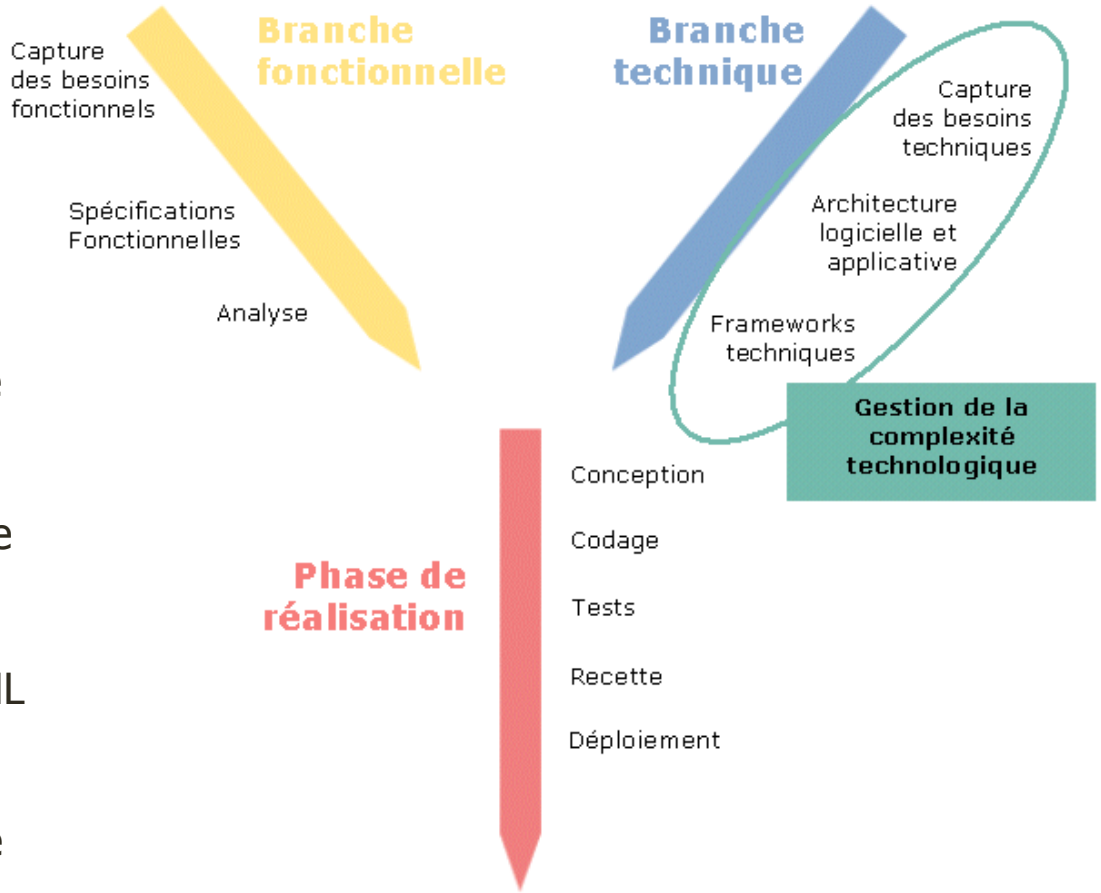
Modèle en spirale (suite)

- Couplage de la nature itérative du prototypage avec les aspects systématiques et contrôlés du modèle en cascade
- Les premières itérations peuvent être des modèles *sur papier* ou des prototypes
- Utilisation possible tout au long de la vie du produit

☞ *Réduit les risques si bien appliqué*

☞ *Les augmentent considérablement si le contrôle faiblit*

2TUP : *Two Track Unified Process*



- S'articule autour de l'architecture
- Propose un cycle de développement en Y
- Détaillé dans « UML en action »
- pour des projets de toutes tailles

Agilité ?

Ce qui NE marche PAS

- Des spécifications complètes en premier

- Commencer par coder sans aucune conception

Ce qui MARCHE

- Principe KISS: Keep It Simple, Stupid !
  - Procéder par incrément

- Impliquer le client
- Classer les tâches par priorité
- Faire des itérations courtes
- Utiliser des tests unitaires

Manifeste agile…

- Idées de base
  - Un produit ne peut pas être entièrement spécifié au départ
  - L'économie est trop dynamique: l'adaptation du processus s'impose.
  - Accepter les changements d'exigences, c'est donner un avantage compétitif au client

- Privilégier :
  - **L'interaction entre les personnes plutôt que les processus et les outils**
  - **le logiciel fonctionnel plutôt que la documentation pléthorique**
  - **la collaboration avec le client plutôt que la négociation de contrats**
  - **la réaction au changement plutôt que le suivi 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

Plusieurs méthodes agiles

- XP : eXtreme Programming
  - Kent Beck, 1999 : focus sur la construction
- SCRUM : mêlée en rugby
  - Met l'accent sur la pratique des réunions quotidiennes
- RUP : Rational Unified Process
  - Vue globale, pour les gros projets
- …

eXtreme Programming

- Maximiser les activités qui apportent une réelle valeur au **logiciel**

- Idées reçues :
  - XP, c'est pour les Développeurs «geeks »
  - XP, cela ne fonctionne que sur de petits projets sans enjeu
  - XP, c'est du TDD et du pair-programming, c'est à dire 2 personnes pour faire le boulot d'une seule...
  - XP, on n'écrit aucune doc
  - XP c'est anti qualité
  - XP c'est impossible dans un domaine régulé (aéronautique)

XP: problèmes ciblés

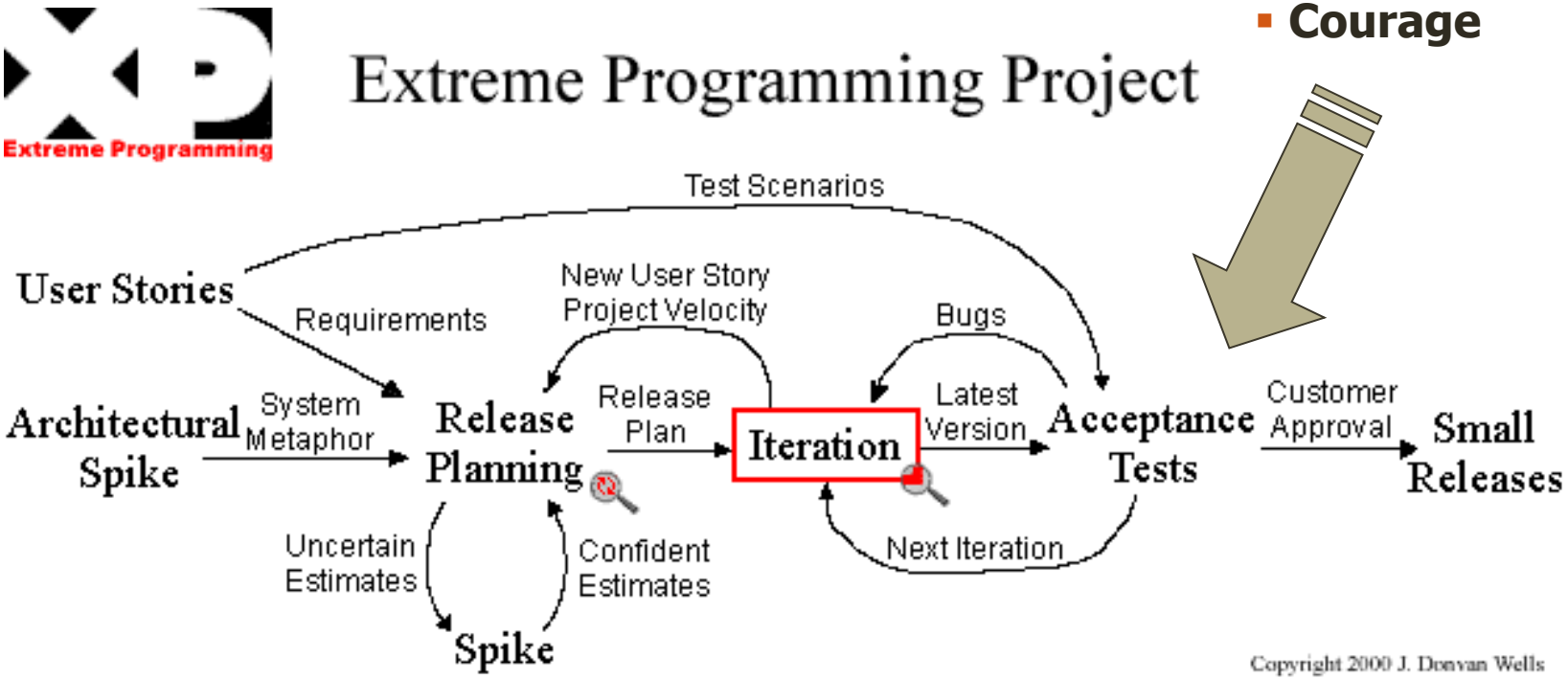
- Défauts
- Travail non terminé
- Dérives
  - (fonctionnalités jamais utilisées)
- Multiplicité des tâches
- Documentation pas à jour
- Obsolescence

eXtreme Programming (XP…) : mise en oeuvre…

- Ensemble de « Bests Practices » de développement
  (travail en équipes, transfert de compétences…)
- plutôt pour des projets de moins de 10 personnes

4 Valeurs

- **Communication**
- **Simplicité**
- **Feedback**
- **Courage**



Communication verbale et équipe colocalisée

- Contacts réguliers avec le client
- User stories
- Mêlée journalière
- Programmation en paires (pair programming)
- Tests Unitaires
- Expérimenter

Feedback, simplicité et courage !

- Consulter régulièrement le client
- Tester vos codes

- Quelle est la chose la plus simple qui est utile ? -> Construit là !
- You Ain't Gonna Need It (YAGNI)
- Absolute simplicity isn't always the best solution, so be as simple as possible but not too simple

- Ayez le courage de corriger, jeter, essayer !!!

Pratiques XP

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests

User stories + Planning Game + Short Cycles

- User stories = récits d'utilisateur
  - Une ou deux phrases résument ce que veut l'utilisateur
  - Décrit comment le système est censé travailler
  - "En tant que recruteur, je peux déposer des offres d'emploi"
- Planning Game = consensus client/développeurs
  - Priorité : compromis entre valeur ajoutée des user stories et complexité de développement
- Short cycles = production rapide de versions limitées de l'application
  - Livraisons fréquentes (feedback immédiat)

Simple Design + Refactor mercilessly

- Simple Design ?
  - Simple mais pas simpliste
  - Focalisé sur les besoins actuels dans l'ordre de leurs priorités
  - Planification par la valeur ajoutée, petites livraisons

- Refactoring
  - Continuellement et progressivement pour gérer la détérioration
  - Refactoring en équipe sans modification du comportement (validé par les tests)

Pair programming + Coding standards

- Pair programming = 2, côte à côte
  - Utilisent le même ordinateur (ou la même feuille de papier)
  - Réfléchissent sur le même code, test, design
  - Au pire 15% de temps en plus, mais 15% de bugs en moins
- Coding Standards
  - Appropriation collective de l'application
  - Facilite la communication

TDD + continuous integration + Acceptance Tests

- Test-driven development
  - Ecrire le test avant le code (Rouge / Vert / Refactor)
  - Le logiciel est testé et validé en permanence
- Intégration continue
  - Assemblage journalier des codes
  - Stabilisation systématique
- Acceptance Tests = tests fonctionnels
  - Définit avec le client
  - Vérification **automatique** de chacun des fonctionnalités

Whole Team + Collective Ownership + Sustainable Pace

- Whole Team
  - Le client n'est pas celui qui paye, mais celui qui utilise
  - Il est toujours disponible pour répondre aux questions
- Collective Ownership
  - Chacun connaît l'intégralité du code !
  - Réorganisation fréquente des binômes
- Sustainable Pace
  - Planification sans heures sup

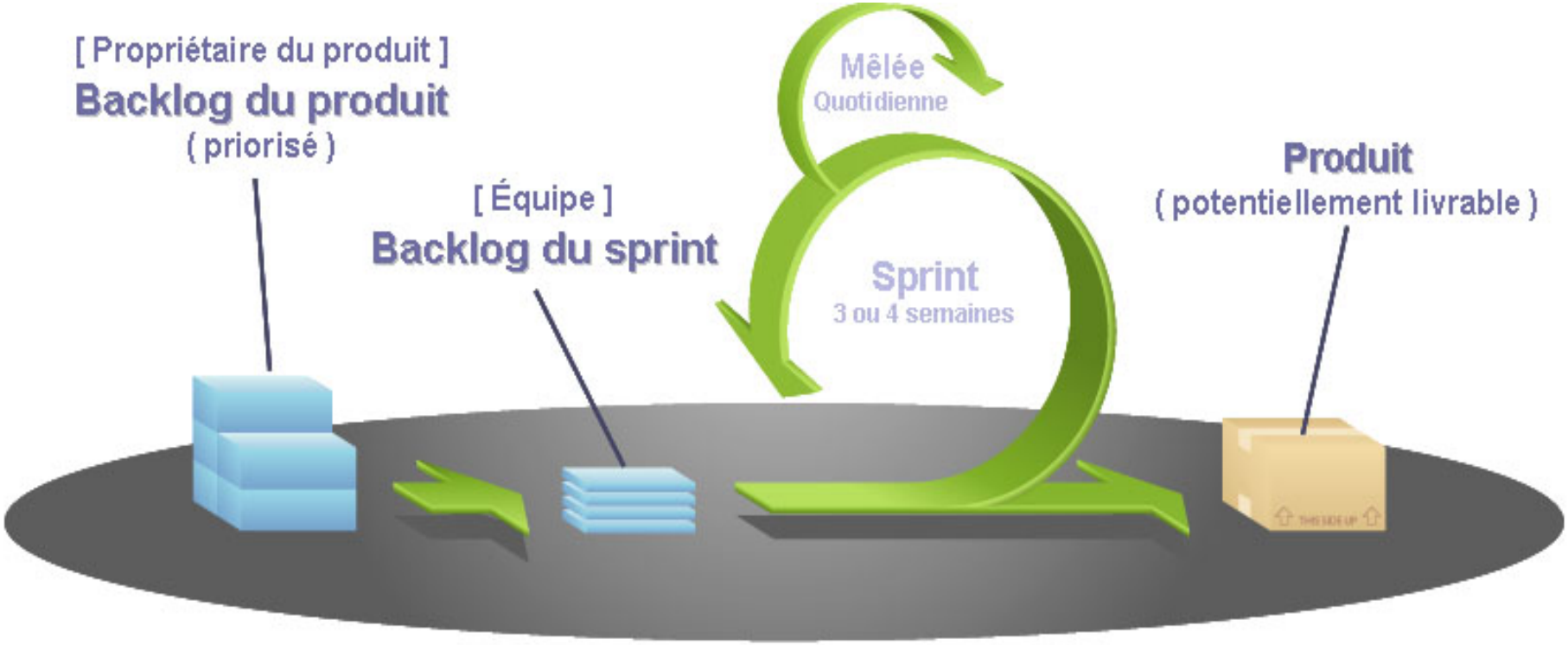
Scrum

- Scrum est un processus agile qui vise à produire la plus grande valeur métier dans la durée la plus courte.
- Un logiciel qui fonctionne est produit à chaque sprint (toutes les 2 à 4 semaines).
- Le métier définit les priorités.
- L'équipe s'organise elle-même pour déterminer la meilleure façon de répondre aux exigences les plus prioritaires.
- A chaque fin de sprint, tout le monde peut voir fonctionner le produit courant et décider soit de le livrer dans l'état, soit de continuer à l'améliorer pendant un sprint supplémentaire.

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.

Processus Scrum



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

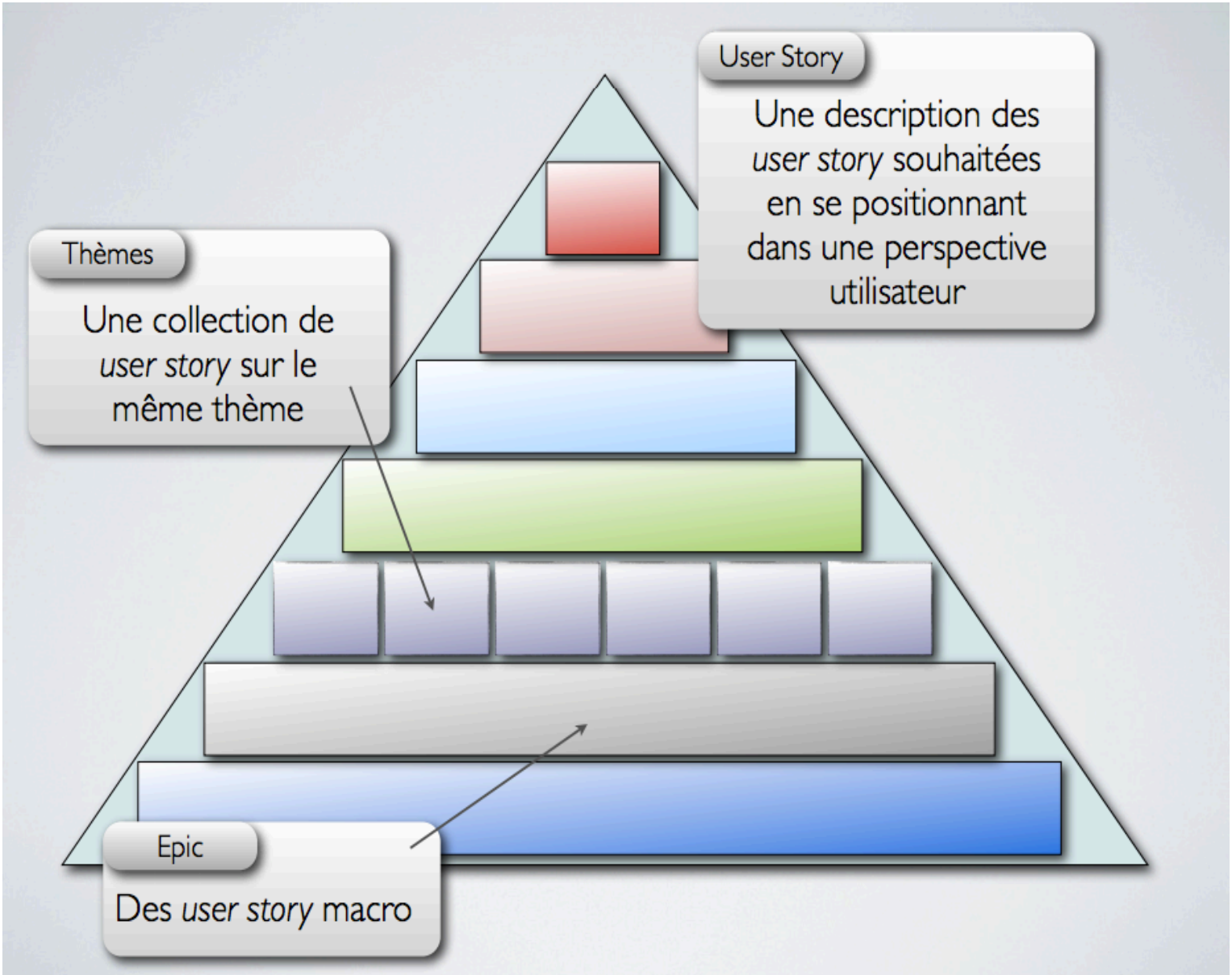
- Product Backlog
  - état courant des tâches à accomplir
- Effort Estimation
  - permanente, sur les entrées du backlog
- Sprint
  - itération de 30 jours
- Sprint Planning Meeting
  - réunion de décision des objectifs du prochain sprint et de la manière de les implémenter
- Sprint Backlog
  - Product 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

Backlog du produit

- Les exigences du produit
- Toutes (Idées, fonctionnalités, Epic, Thème, etc.)
- Exprimées en User Stories
- Le PO le maintient organisé
- Toujours estimé et avec les priorités

Université Nice Sophia Antipolis

User Stories, Epic…



SPRINT

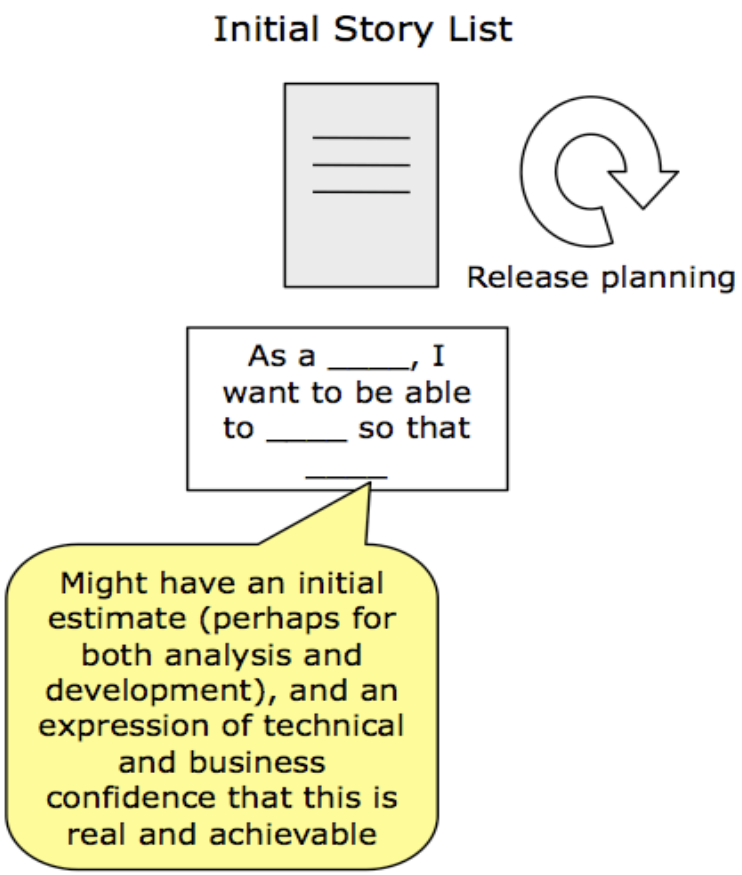
RELEASE

FUTUR

User Story

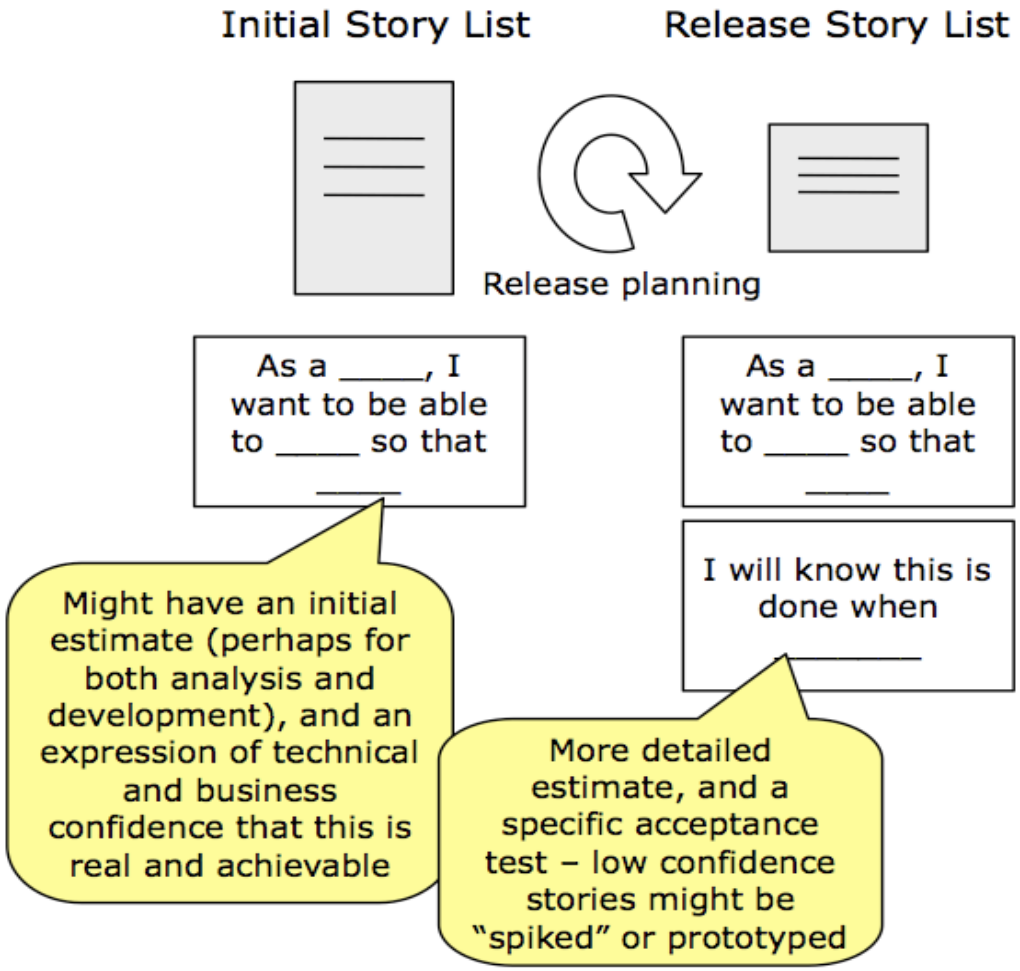
- Une ou deux phrases résument ce que veut l'utilisateur
- Décrit comment le système est sensé travailler
- Ecrite sur une carte
- Contient suffisamment de détails pour pouvoir être estimée

User story



- Au Recto :
  - Fonctionnalité exigée sous la forme d'un «récit utilisateur»,
  - Sa priorité attribuée par l'utilisateur et
  - Le temps estimé par l'équipe pour sa réalisation.

User story



Au Verso :

- conditions d'acceptation définies par l'utilisateur
- niveau de risque (optionnel)
- modifications de la demande (optionnel)

Bonne « story » : quelle taille ?

- Des petites user stories pour un futur proche
- Macro (Epic) pour les prochaines
- Les user stories sont progressivement affinées dans le temps, plus elles s'approchent de la fin
- Deux types de grandes user stories
  - Les user stories complexes : intrinsèquement grande et sans possibilités de les réduire
  - Les user stories combinées : Plusieurs user stories combinées en une seule

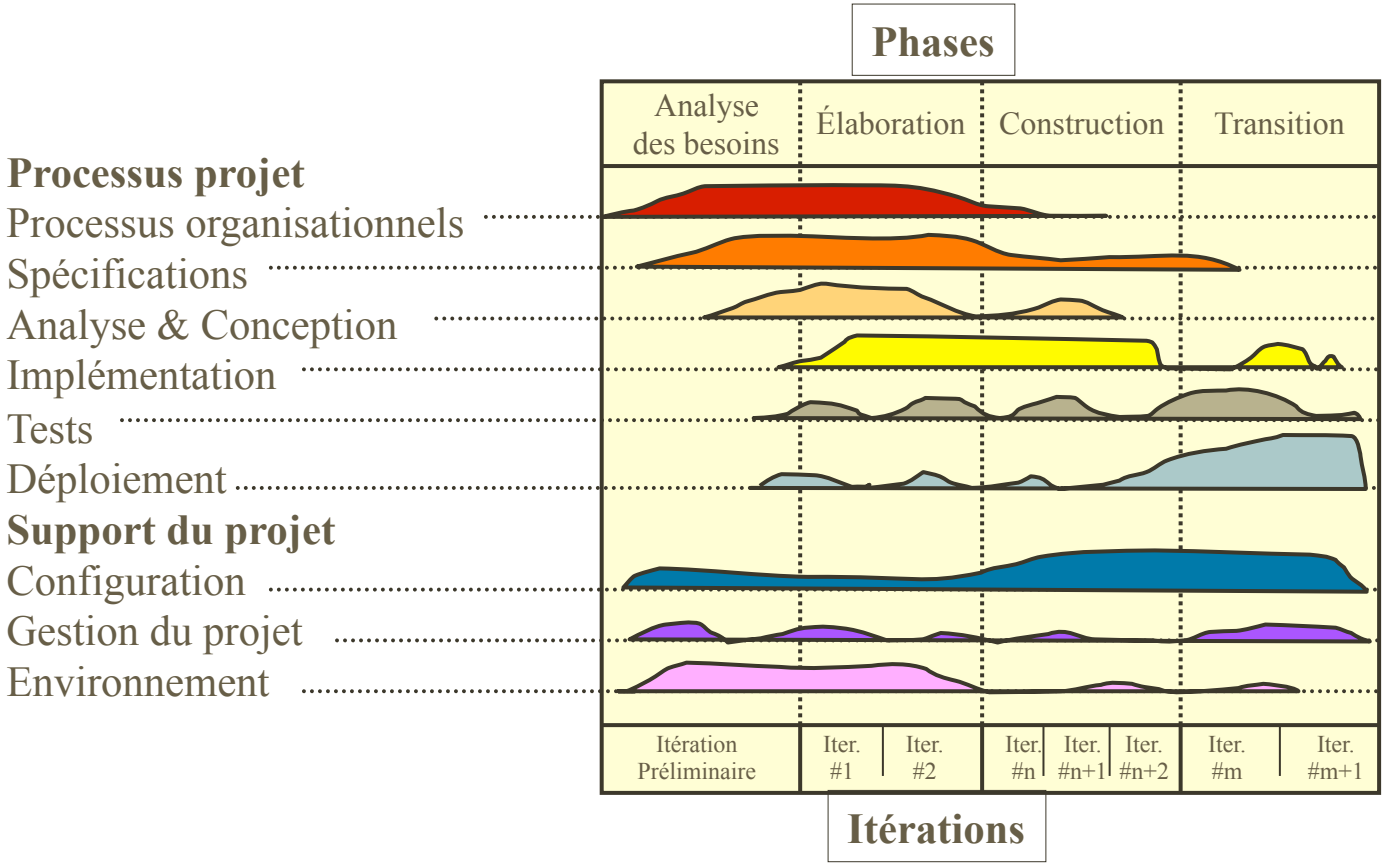
Bonne « story » : testable ?

SMART

- **Spécifique** - défini et explicite
- **Mesurable** - quantifiable et mesurable
- **Atteignable** - qui peut être réalisé et validé
- **Relevant** - pertinent pour la story
- **Temporaire** - limité dans le temps

- Facilite la rédaction des critères d'acceptation

RUP : *Rational Unified Process*



- Promu par Rational
- Le RUP est à la fois une méthodologie et un outil prêt à l'emploi (documents types partagés dans un référentiel Web)
- plutôt pour des projets de plus de 10 personnes