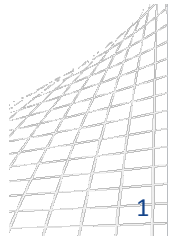
Introduction à l'Agile



D'où je viens

- Études
  - M2 info : Paris Diderot (2009)
  - MS Management de Projets Technologiques : ESSEC / Telecom Paris (2010)
- Aujourd'hui
  - Consultant à OCTO Technology (Conseil en SI)

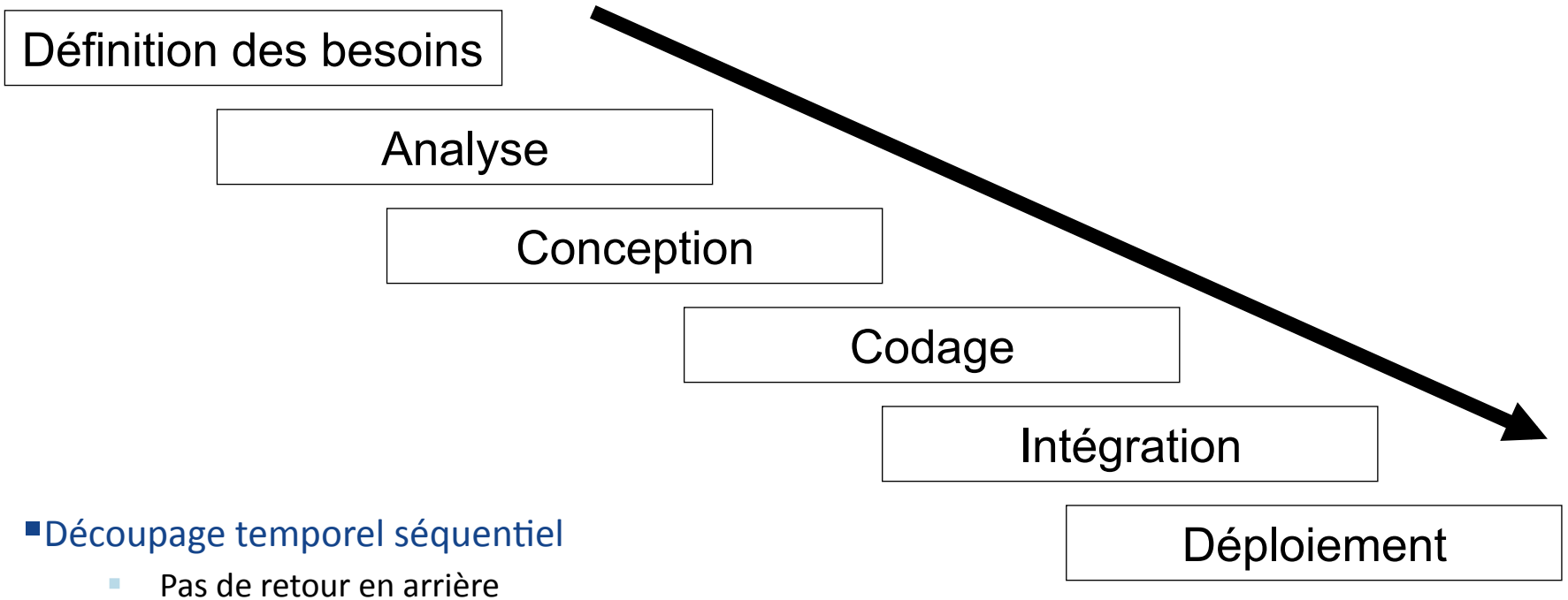
Ce que je souhaite montrer avec cette présentation

- Donner un aperçu de ce qu'est la gestion de projet « Agile »
- Faire un retour sur mes expériences passées sur des projets agiles et d'autres

Ce que j'ai fait en termes de projets jusqu'ici

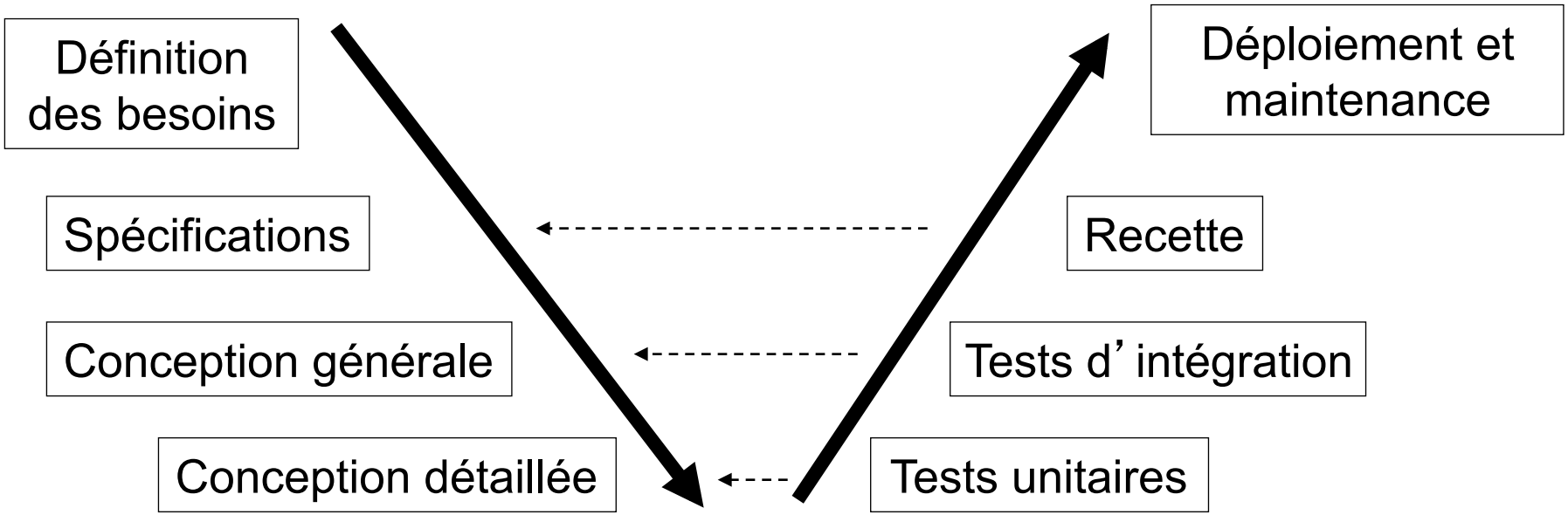
- Projet en M1 pour une société privée
- Projets de fin d'études (M2 et MS)
- Projets « de la vraie vie » pour OCTO

Ce qu'on trouve souvent sur les projets (1/2)
La cascade



- Découpage temporel séquentiel
  - Pas de retour en arrière
    - Suppose que tous les besoins sont définis en début de projet
    - Suppose que la conception soit « bonne dès le départ »
  - Retarde la résolution des risques (intégration)
- Chaque étape est formellement validée
  - Projet focalisé sur les documents et les réunions de validation

Le cycle en « V » en 2 minutes

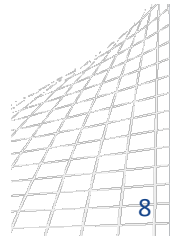


- Chaque étape de la branche de droite définit les critères d'appréciation et d'acceptation des étapes de la branche de droite

En bref, l'Agile c'est :

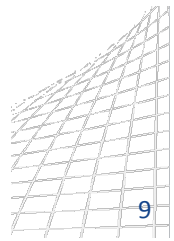
Un corpus d'outils et concepts issus de sources diverses

- Le terme « Agile » est apparu aux alentours de 2000
- C'est un recueil de bonnes pratiques et outils issus du monde industriel (pre-2000 la plupart du temps) adaptées au développement logiciel
- L' « Agile » s'est inspirée de plusieurs méthodes dont
  - Le Lean management
  - Le Toyota Production System



Une gestion de projet soutenable qui permet de progressivement produire un logiciel

- Itératif :
  - Entre 1 semaine et 1 mois en général
  - À chaque itération, c'est un cycle complet de développement qui redémarre (spec, dev, recette)
- Incrémental :
  - Les fonctionnalités à implémenter sont priorisées entre elles
  - À chaque fin d'itération le logiciel produit est fonctionnel
- Théoriquement, l'équipe doit être capable d'enchaîner les itérations sans s'arrêter



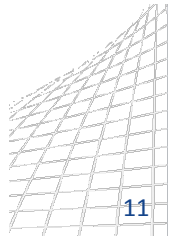
Think BIG

Start small

Deliver *Quickly*

Des principes

- Itérations
- Planification long terme
- Amélioration continue
- Développement piloté par les tests
- Qualité du code
- Binômage



Une méthodologie « ad-hoc »

- Pilotage du projet à la valeur métier
  - La priorisation en fonction des demandes métier
  - Effet « Stop ou encore » / « Stop the line »
- Possibilité d'introduire des modifications à tout moment
  - Gestion des risques projet incluse dans la démarche
  - Permet de travailler en contexte incertain
  - Le changement n'est pas nécessairement néfaste
  - « Fail fast »

Que peut apporter l'Agile ?

- Lutter contre un contexte incertain
- Limiter le coût des défauts
  - Le coût d'un défaut augmente exponentiellement en fonction du temps mis à le détecter. Plus on met le doigt dessus tôt, plus il est peu coûteux (techniquement et financièrement) de le corriger.
- Mettre en place un processus d'amélioration continue (*Kaizen*)
  - Les outils généralement utilisés, et la philosophie prônée empêchent les gens de rester dans leur coin

Un peu de vocabulaire

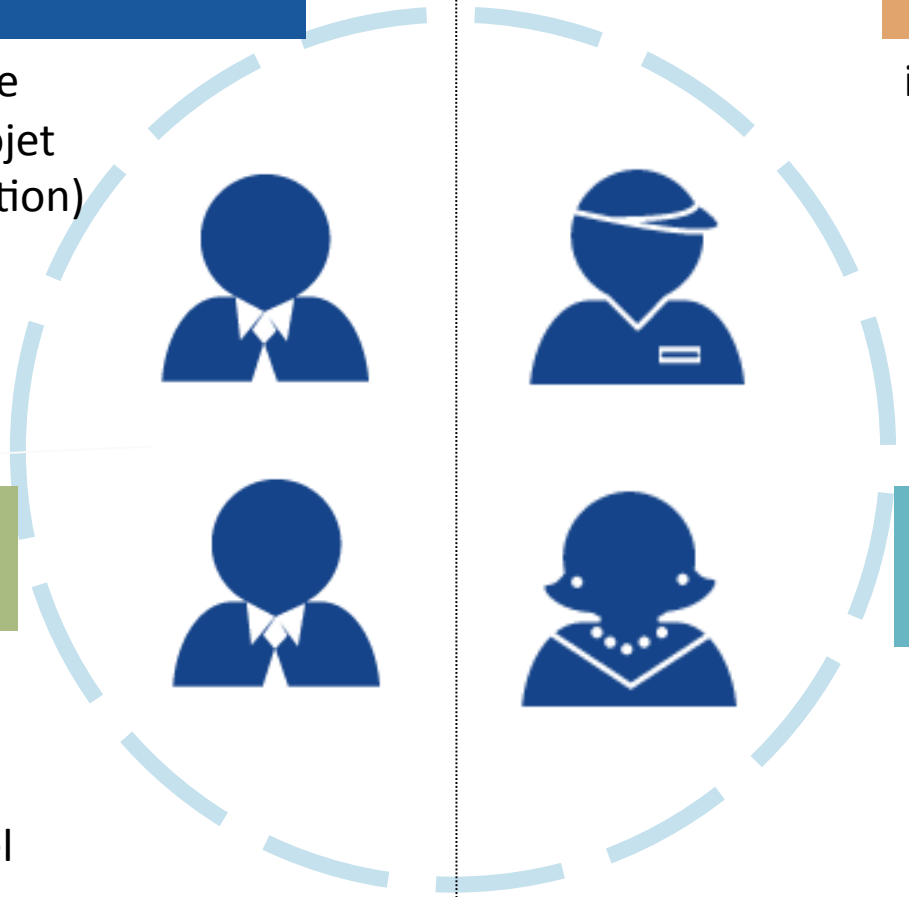
Les membres « type » d'une équipe

Responsable Produit

assure le pilotage fonctionnel du projet (exigences et validation)

Développeurs

implémentent le logiciel dans les standards de qualité



Utilisateurs

expriment les besoins métier et évaluent l'utilisation du logiciel

Coach

aide l'équipe à s'approprier la méthode et à s'améliorer

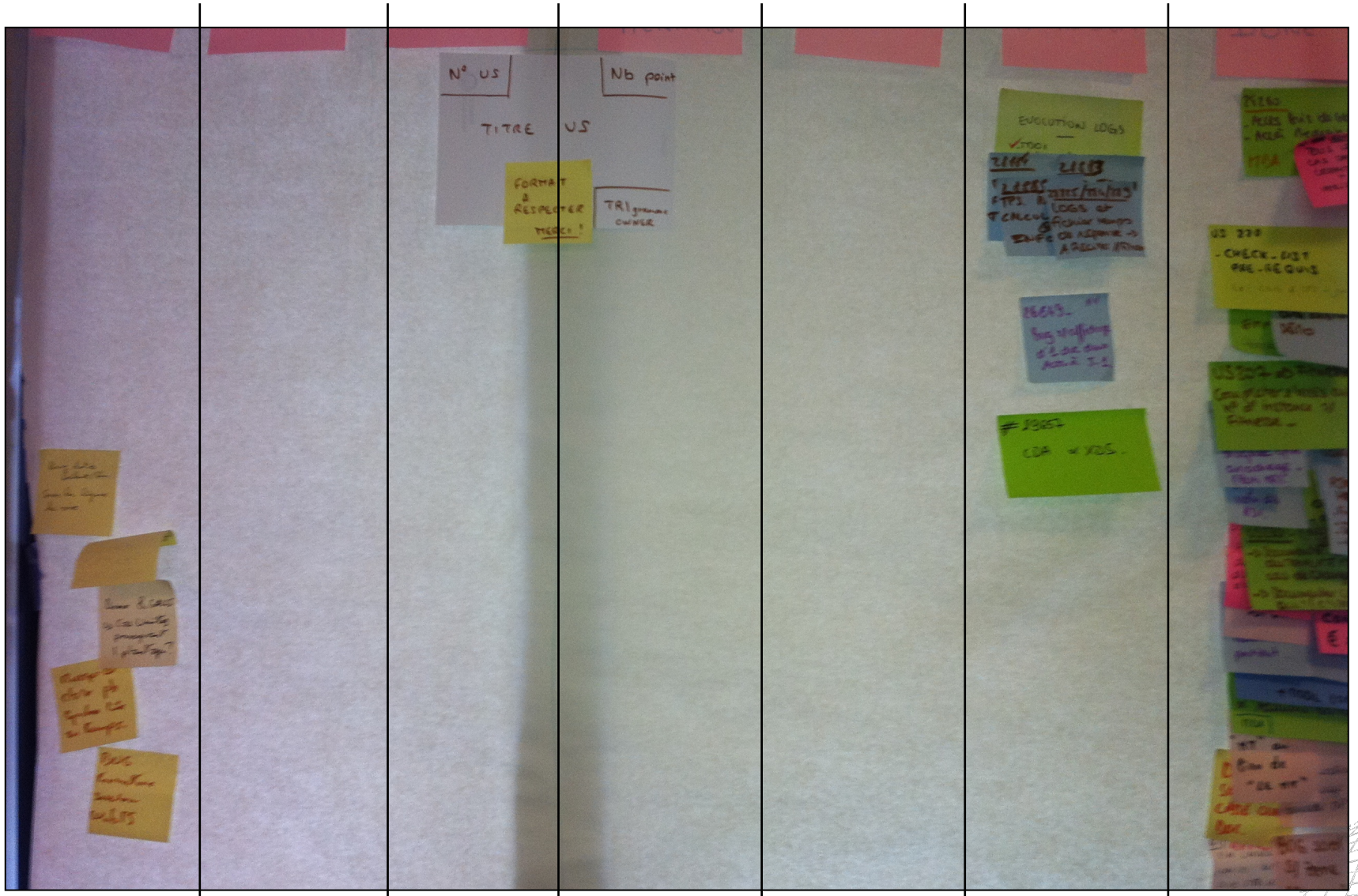
« Clients » du projet

Équipe qui réalise le projet au quotidien

Les outils les plus utilisés

- User Story / Complexité
- Planning Game
- Backlog
- Kanban
- Stand-up meeting

Kanban



Qu’est-ce qui va aider mon projet Agile ?

- Équipes de petite taille (10 pers.)
- Vocabulaire commun
- « *Gemba* »
- Disponibilité du responsable produit / des utilisateurs
- Rigueur (mais pas rigidité)

Rigueur

- Se tenir aux pratiques que l'on choisit d'utiliser, ne rien laisser filer
- « Chacun son rôle »

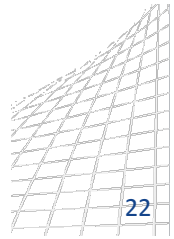
Les choses pouvant mettre en difficulté le projet

- Les sceptiques de la méthodologie
- Éclatement géographique de l'équipe
- Responsable produit « releveur de compteurs »
- Conditions de validation pas claires

A-t-on besoin de tous les outils / processus tout le temps ?

La gestion de projet « Agile » c'est avant tout une grosse boîte à outils

- On prend les outils dont on a besoin
- On évite ceux qui ne servent pas
- Il y a des outils que l'on gagne toujours à embarquer dans un projet Agile (US, Backlog, TDD …). Ils sont en général non négociables.



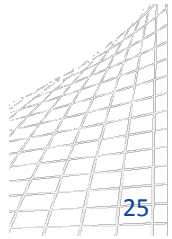
Maturité de l'équipe

- On ne donne pas les ciseaux pointus au petit dernier tout de suite, à tous les coups il va se blesser.
- Certains outils ne sont pas pertinents immédiatement
- « *La vraie maîtrise, c'est de réussir à s'approprier assez les concepts Agiles pour pouvoir les plier à ses besoins projet* » (Jim Highsmith)

Retour d’expérience : Mon projet « de la vraie vie » de l’an passé

Contexte

- 3 gros livrables, l'équipe a changé à chaque fois
- Emplois du temps de certains intervenants pas simples
- Responsable produit avec une implication très variable
- Évolutions techniques profondes non prévues à mener en plein milieu de projet
- UDD + tests unitaires



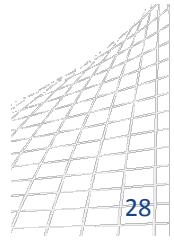
Bilan du projet

- Pilotage à la valeur : Conformité du logiciel aux attentes
- Communication : Transparence sur la réalisation (Kanban, Redmine …)
- Propriété collective
- Régressions : Peu de régressions grâce aux tests unitaires

Retour d'expérience : Mon stage de fin de M2

Contexte

- Startup
- Réunions de l'équipe au complet toutes les 3 semaines environ
- Rétroplanning pour suivre / projeter l'avancement
- Chaque développeur est responsable d'une partie de la plateforme
- Recette exclusivement manuelle



Problèmes rencontrés

- Régressions
- Vision projet peu transmise en dehors de l'essentiel
- Conditions de validation des fonctionnalités pas claires
- Peu de propriété collective du code
- Chacun dans son silo

Quels outils « Agiles » auraient pu être utiles ?

- Régressions : TDD / ATDD
- Vision projet : Meilleure communication du PO sur ses attentes, implication accrue pendant la recette
- Conditions de validation : Meilleure formalisation / meilleur découpage des fonctionnalités demandées / recettes plus régulières
- Propriété collective du code : Pair programming / revues de code
- Manque de visibilité sur l'activité des autres : Kanban

Un projet apparemment en péril n'est pas une fatalité !

- Ne pas laisser filer les pratiques existantes
- Changer l'organisation là où cela est nécessaire (ajouter / enlever des outils ou des processus).
- Il faut mettre le doigt là ou ça fait mal, faire ce qu'il faut pour que cela change et s'y tenir

Merci !

ANNEXES

Un Manifeste pour le développement Agile

« Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :

- **Les individus et leurs interactions plus que les processus et les outils**
- **Des logiciels opérationnels plus qu'une documentation exhaustive**
- **La collaboration avec les clients plus que la négociation contractuelle**
- **L'adaptation au changement plus que le suivi d'un plan**

Nous reconnaissons la valeur des seconds éléments mais privilégions les premiers ».

http://agilemanifesto/iso/fr

Le coût du défaut

- Plus un défaut est détecté tard dans un projet (par rapport à son introduction), plus il est difficile et coûteux de le corriger
- Le coût de correction de ce défaut augmente :
  - Exponentiellement avec le temps mis pour le découvrir avec une gestion de projet de type cycle en 'V'
  - Logarithmiquement (en théorie, en fait pas tout à fait) avec une approche par cycles courts (comme l'Agile)

Test Driven Development (TDD)

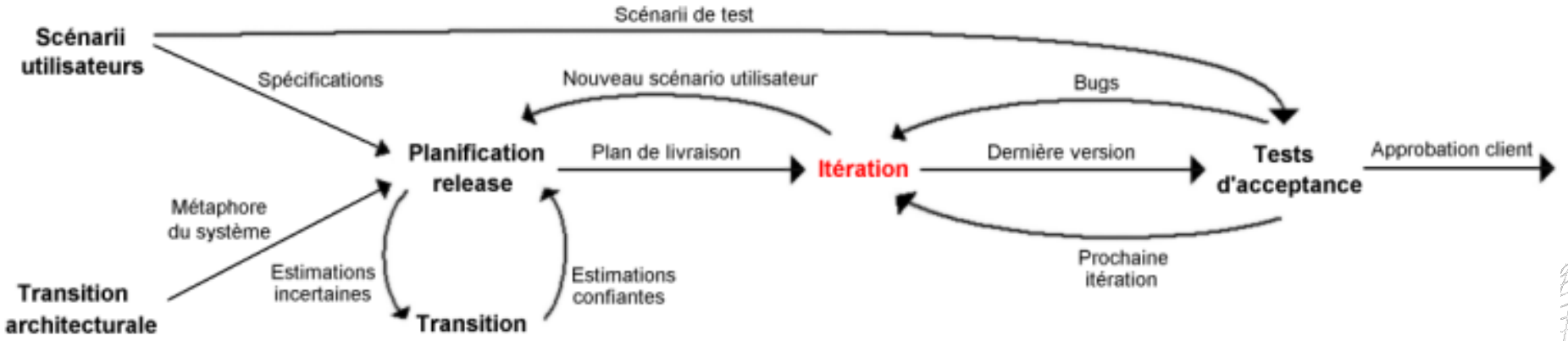
- « Développement piloté par les tests » en français. C'est une méthode de développement logiciel qui consiste à commencer par écrire les tests unitaires avant le code à proprement parler.
- Le TDD a pour objectif de
  - Ne pas laisser passer de cas de test (trivial ou pas)
  - N'écrire que le code nécessaire au fonctionnement de l'application
  - Écrire du code « propre » (pas de duplication, des petites fonctions, pas d'effets de bord …)
  - Mettre en place et entretenir en continu le « harnais » de tests unitaires de l'application (éviter le syndrôme « Bref, je remets tout à demain »)
- On retrouve le TDD dans la plupart des pratiques de développement logiciel Agile (comme eXtreme Programming)

Acceptance Test Driven Development (ATDD)

- « Pilotage par les tests métier » en français
- Permet à la MOA (Maîtrise d'OuvrAge), de mettre en place des tests pour vérifier la validité « métier » d'une implémentation
- Outils populaires :
  - FitNesse (www.fitnesse.org)
  - Twist (http://www.thoughtworks-studios.com/agile-test-automation)
- Idée de base derrière ces outils :
  - Mise à disposition de la MOA d'une sorte de wiki où ils peuvent écrire leurs spécifications « métier » selon un certain formalisme
  - Les développeurs codent des « fixtures » pour permettre à l'outil d'interagir avec l'application et de tester les comportements souhaités

eXtreme Programming

- En informatique et plus particulièrement en génie logiciel, Extreme Programming (XP) est une méthode agile de gestion de projet informatique adaptée aux équipes réduites avec des besoins changeants. Elle pousse à l'extrême des principes simples.



Backlog

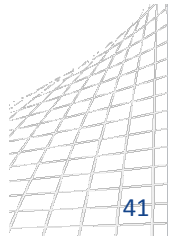
- Sorte de grande liste des choses à faire sur un projet
- C'est un outil de pilotage de projet (calculs de vélocité …)
- Il est utilisé pour alimenter le Kanban board
- Chaque User Story y est répertoriée avec diverses de ses caractéristiques :
  - Priorité par rapport aux autres
  - Description succinte
  - Complexité estimée
  - Numéro de l'itération dans laquelle elle est embarquée
  - Statut en fin d'itération (OK / KO)

Kanban Board

- Tableau découpé en silos d'activités
- Toutes les User Stories réalisées passent dedans
- Chaque métier prend dans un silo pour en alimenter un autre
- Fonctionnement en flux poussé
- Inspiré des jeux de cartes (Kanban) des usines Toyota

User Story

- Fonctionnalité « élémentaire » à développer
- Ne peut pas être redécoupée (dans l'idéal)
- Assez petite pour tenir dans une itération



Itération

- Durée de temps pendant laquelle le développement se fait
- Se termine par une démonstration permettant de valider les User Stories réalisées
- Caractérisée par le fait que l'équipe de développement s'engage à réaliser un certain nombre de User Stories du backlog (en formant un backlog d'itération), et que personne ne peut modifier ce dernier une fois l'itération commencée)

Planning Game

- Outil d'estimation
- Destiné à toutes les personnes intervenant sur le projet
- Permet de faire discuter MOA et Devs sur les User Stories, de poser les dernières questions sur les règles de gestion à implémenter, et de décider des conditions de validation si besoin
- Le planning game se joue en général en fin d'itération afin de constituer le backlog d'itération de la suivante

Vélocité

- Nombre de points de complexité validés dans une itération
- On en calcule la moyenne glissante pour avoir une estimation plus précise de ce qu'une équipe peut « embarquer » en termes de points de complexité dans son backlog d'itération
- Indicateur exploitable en termes de pilotage après plusieurs itérations (le temps qu'elle se stabilise)