Université de Rennes | IUT de Lannion

R3.10 – Management des Systèmes d'Information

Cycles évolutifs et méthodes Agile

Ressources sous Teams

R3.10

Programme

Ressource R3.10 partie « Cycles évolutifs et méthodes Agile »

- 3h CM – DS en fin de semestre
- 6h TD – User Stories : Expression des besoins en Agile
- 8h TP – 4 ateliers « Serious Game »

… puis application de la méthode Agile Scrum en SAÉ 3 et 4

Cycles évolutifs et méthodes Agile

1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Cycles évolutifs et méthodes Agile

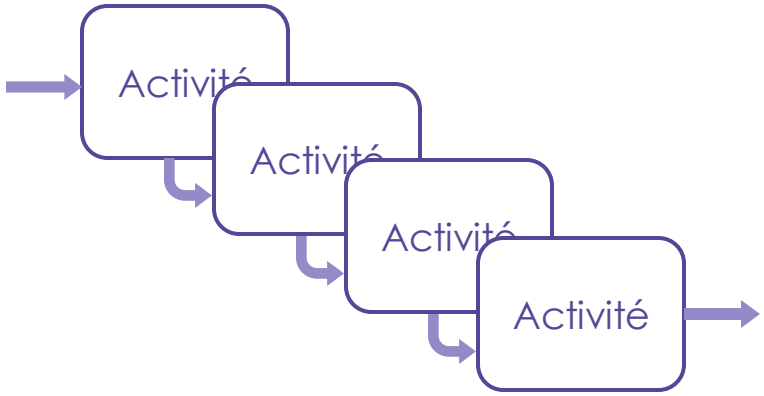
1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Cycle de vie d'un Projet (rappel)

Cycle de vie du projet : façon dont s'enchaînent les activités d'un projet

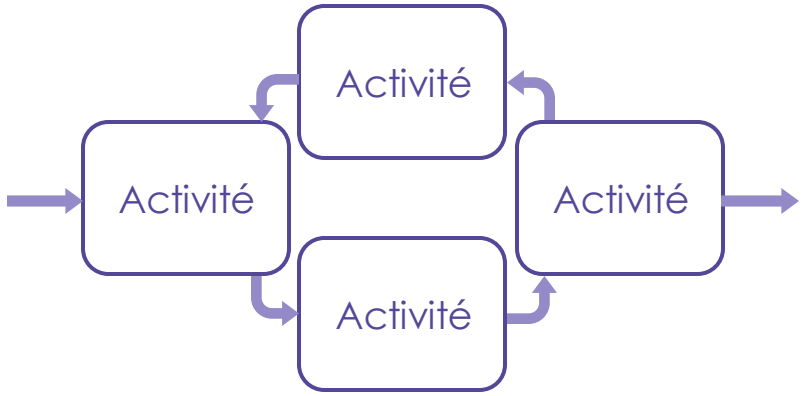
Deux grandes familles :

Les cycles de vie « classiques »



Leurs activités sont séquentielles

Les cycles de vie « évolutifs »

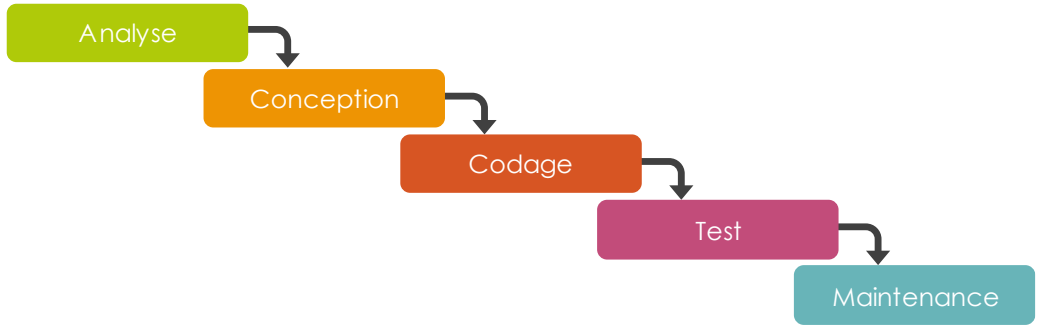


Leurs activités se répètent

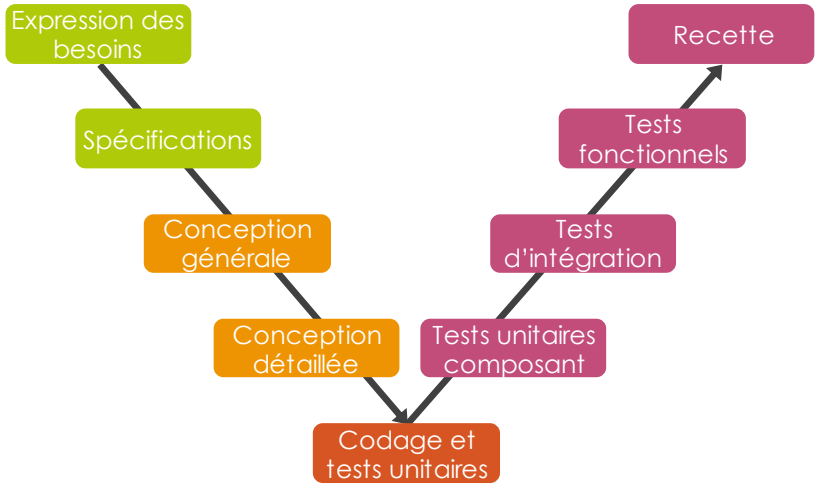
Cycles de vie « classiques » (rappel)

Les plus courants :

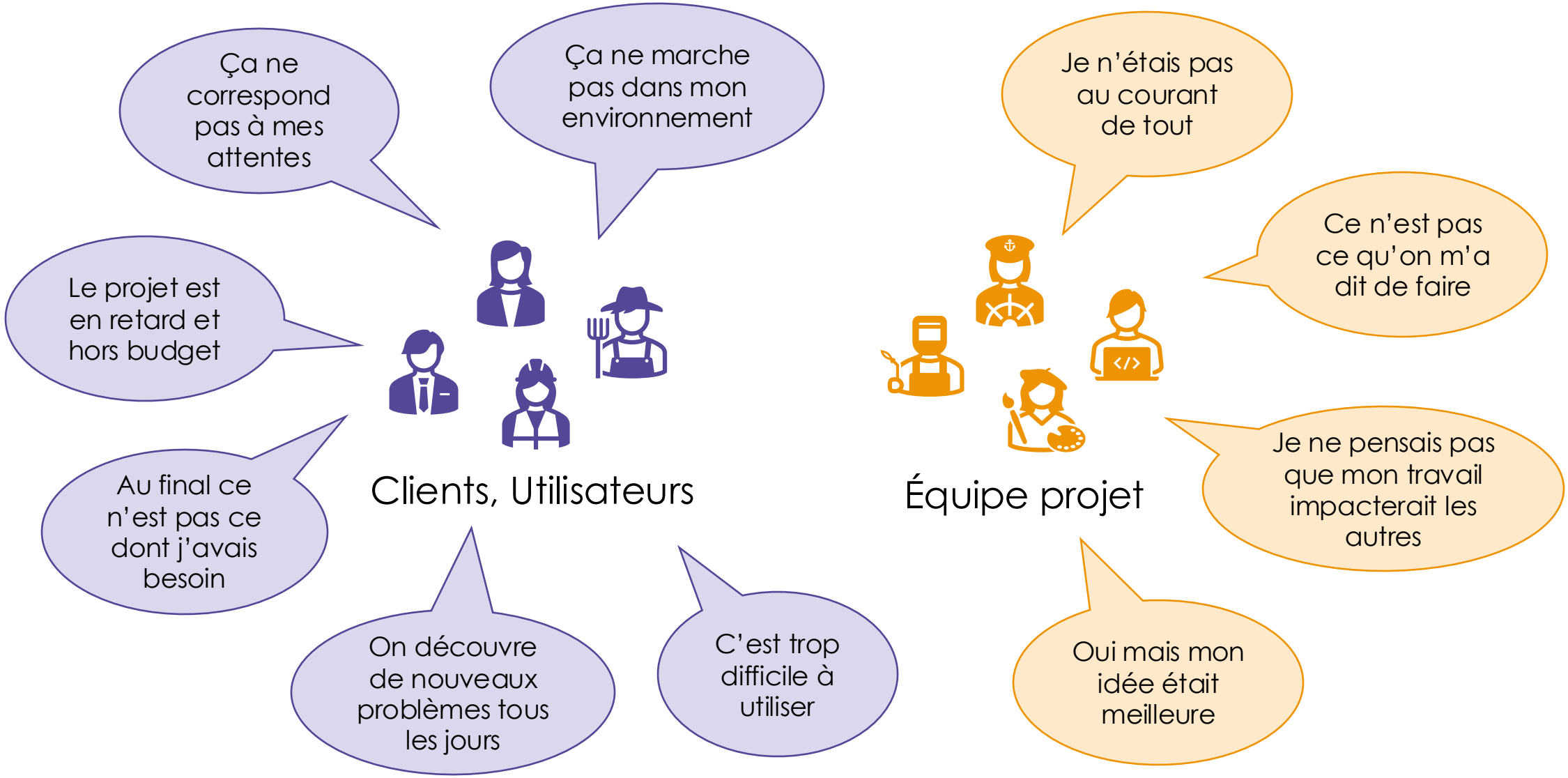
- **Modèle en cascade (Waterfall)**
  - Un des premiers modèles
  - De nombreuses variantes existent



- **Cycle en V**
  - Le plus répandu
  - Adapté aux systèmes complexes
  - Nombreuses étapes de test



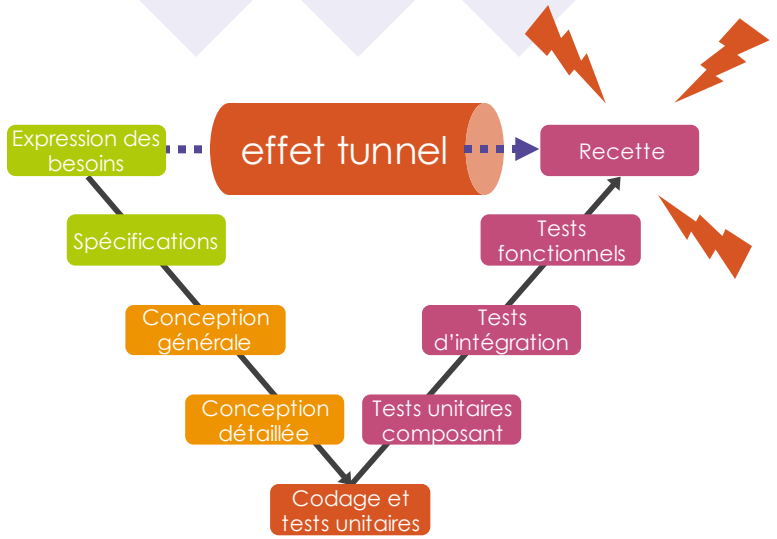
Symptômes des projets en cycles « classiques »

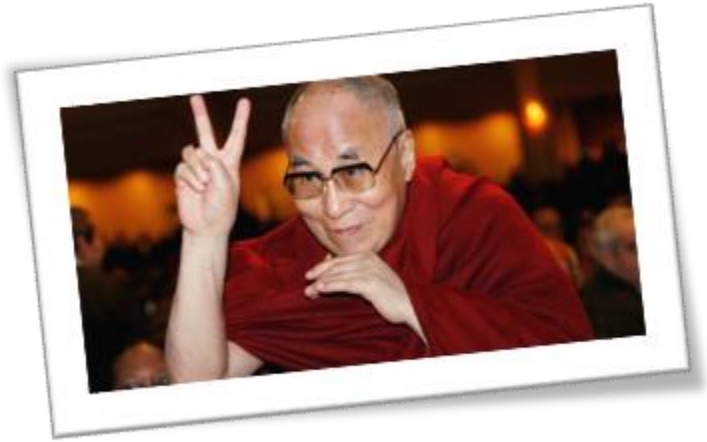


Facteurs d'échec des cycles « classiques »

Ignorer le changement

- L'environnement technique et économique **évolue rapidement**
- Les besoins et les souhaits des **clients changent**
- Les **priorités** du management aussi





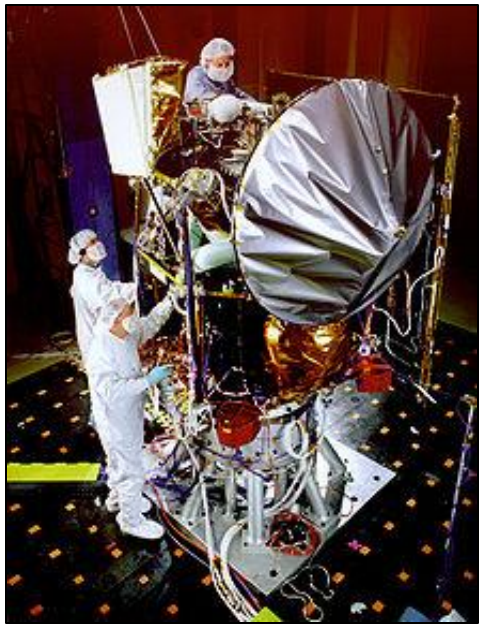
« Au sein de cet environnement instable et turbulent, un seul élément reste constant : le changement. »

Le Dalaï-Lama

Facteurs d'échec des cycles « classiques »

Sous-estimer l'importance de la communication

- Partager un **vocabulaire commun**, une **vision commune** des enjeux, objectifs, et de l'avancement projet



Exemple : la sonde Mars Climate Orbiter

- *Lancée en 12/1998, 7 mois de transit puis manœuvre pour insertion sur orbite martienne*
- *La sonde se place sur une orbite trop basse et se détruit en traversant l'atmosphère*
- *Coût de l'opération : 125 M$!*

Les causes ?

- *Les valeurs de freinage communiquées par LOCKHEED étaient en unités anglo-saxonnes*
- *Mais la NASA attendait des données en système métrique…*

Ci-contre : Un étudiant de R&T assistant un étudiant InfoCom sous l'œil attentif d'un étudiant de MP lors du montage de la sonde…

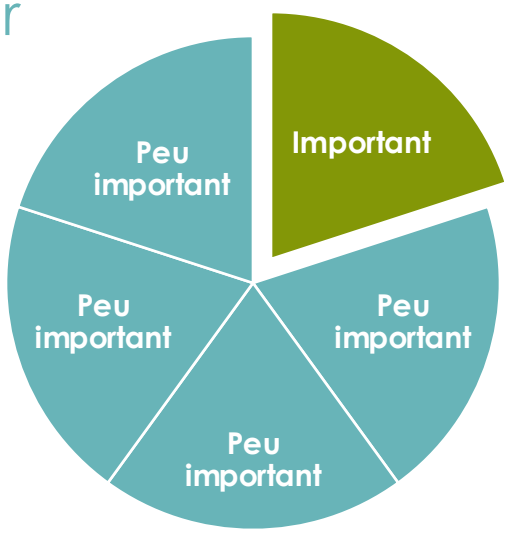
Facteurs d'échec des cycles « classiques »

Ne pas écouter l'utilisateur

- Risque de consacrer un temps précieux à des fonctionnalités **présentant peu d'intérêt pour l'utilisateur final** (peu valeur métier)
- Mais en pratique, on ne peut attendre de tout savoir pour commencer…

80% de la valeur d'un logiciel est portée par 20% des fonctionnalités !

- 4 fonctionnalités sur 5 présentent peu de valeur pour l'utilisateur final



- Ne produire que les plus importantes =
  - Économie de 80% de budget
  - On obtient rapidement un produit
  - Peu de perte de valeur pour l'utilisateur

Cycles évolutifs et méthodes Agile

1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Les cycles de vie « évolutifs »

Développement logiciel = cycle d'activités récurrentes

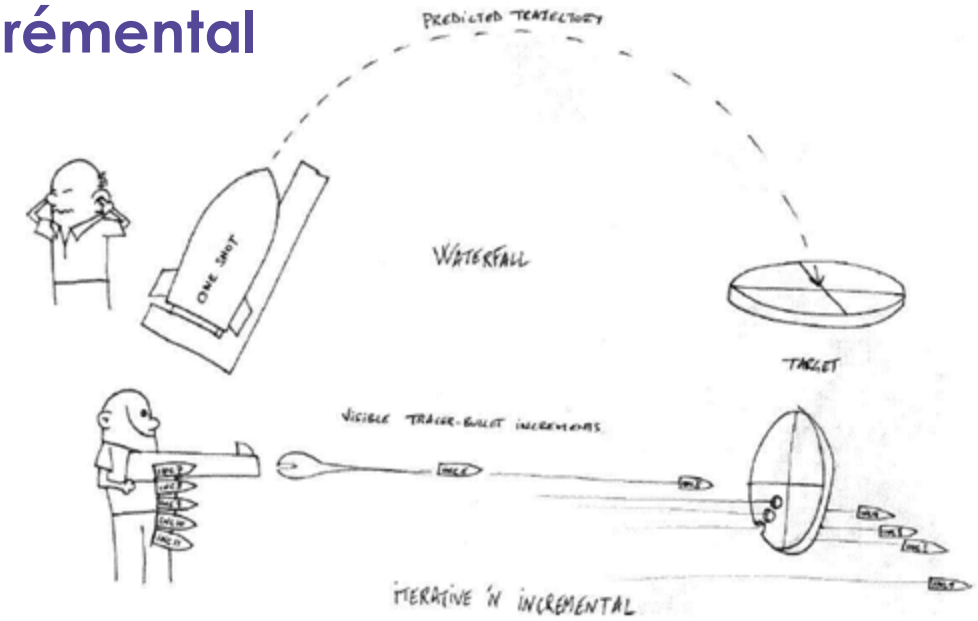
Définition des exigences

Conception de l'architecture

Développement

Intégration et validation

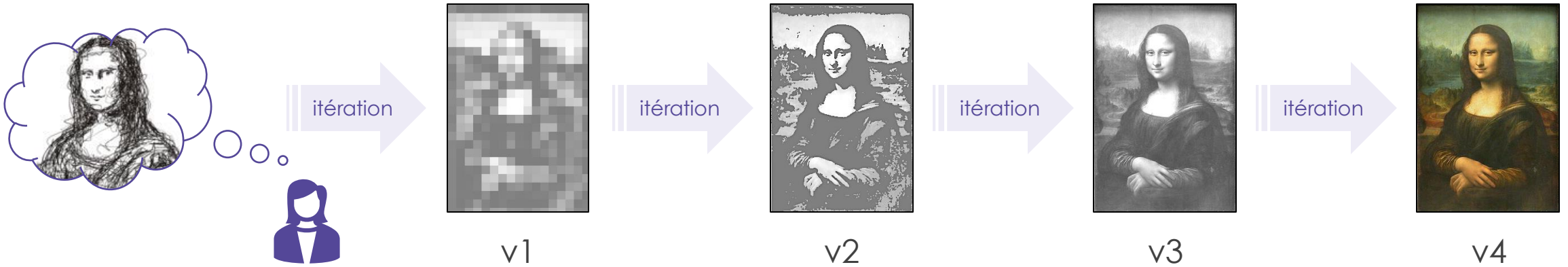
- Le projet est **itératif** et **incrémental**



Produit v1
Produit v2
…
Produit final

Évolutif = itératif et incrémental

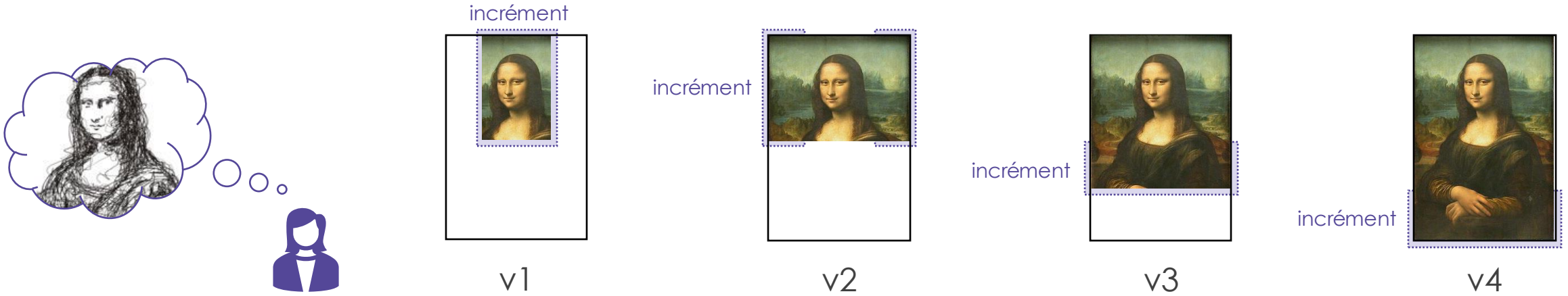
Itératif : construit selon des cycles répétitifs (itérations)



- **Itération** : étape de développement **courte et rapide**
- Tous les besoins ne sont **pas nécessairement définis lors du démarrage**
- **Feedback utilisateur** pour clarifier, améliorer, faire évoluer le produit

Évolutif = itératif et incrémental

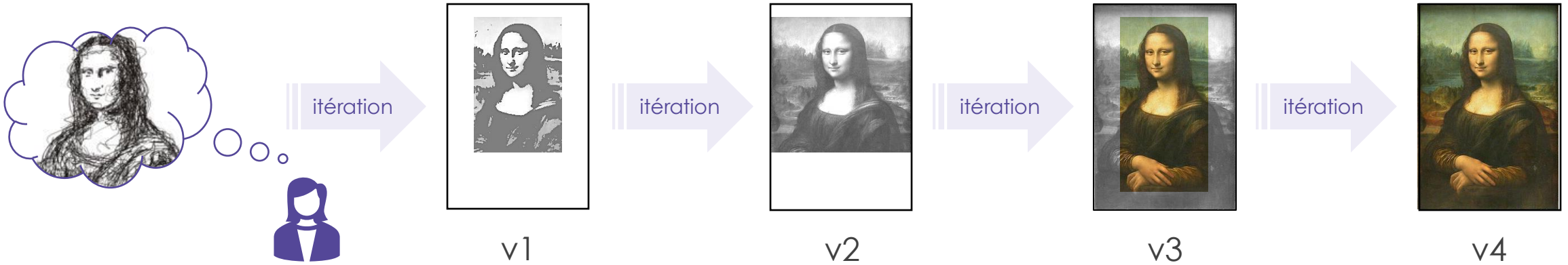
Incrémental : construit par incréments



- **Incrément** : sous-partie **fonctionnelle et cohérente** du logiciel final
- Chaque incrément ajoute de nouvelles fonctionnalités
- Les fonctionnalités (exigences) sont traitées par ordre de priorité

Évolutif = itératif et incrémental

Évolutif : construit par incréments lors d'itérations



- Chaque version est réalisée lors d'une itération → itératif
- Chaque version fournit un produit cohérent → incrémental

→ **Requiert une architecture évolutive et stable**

Bénéfices des cycles « évolutifs »

Une 1ère version est fournie rapidement

- Testable par des « primo » utilisateurs
  → **découverte des problèmes au plus tôt**
- Retour sur investissement **(ROI) visible et rapide**

Risques d'échec diminué

- Parties les plus importantes fournies en premier, donc **testées plus longuement**
- Ajout possible d'exigence à tout moment
- **Développement en parallèle** facilité

Limites des cycles « évolutifs »

Comment découper des exigences en incréments ?

- Trop d'incréments → ingérable
- Trop peu → revient à un cycle « classique »
- **Exige une certaine vision** du produit final

Concevoir une architecture stable et évolutive…

- Exige de nouveau une certaine vision du produit final
- **Ne traite pas toutes les évolutions** (notamment celles qui remettent en cause l'architecture)

Quid de la prise en compte du changement … ?

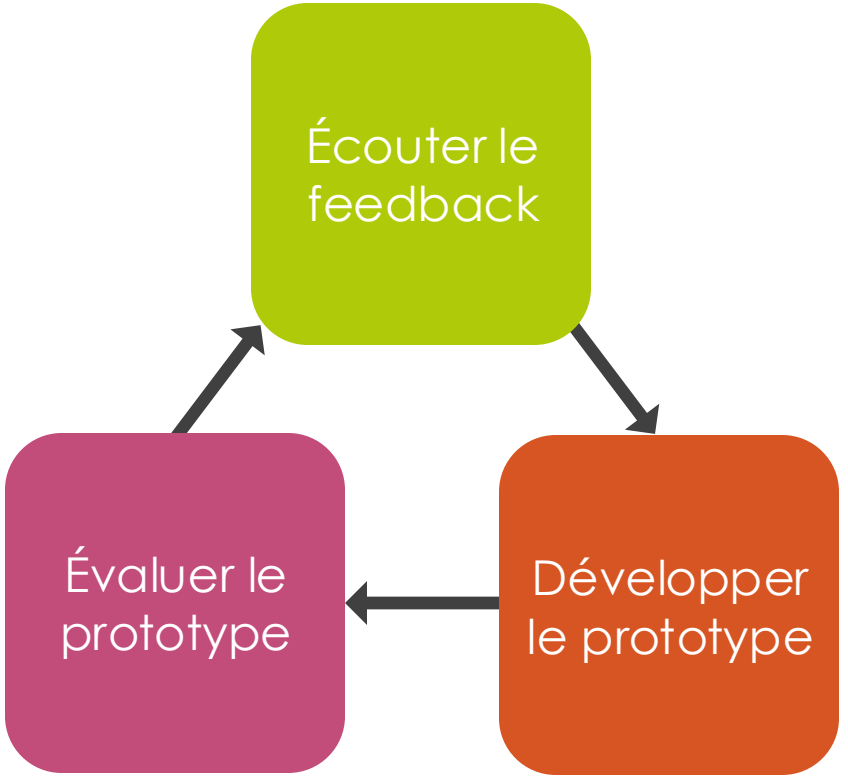
Exemple de cycle « évolutif » : le Prototypage

Produit fait sur plusieurs itérations :

- Les développeurs **construisent un prototype** selon les attentes du client
- Le **prototype est évalué** par le client
- Le client donne son **feedback**

Le cycle se répète :

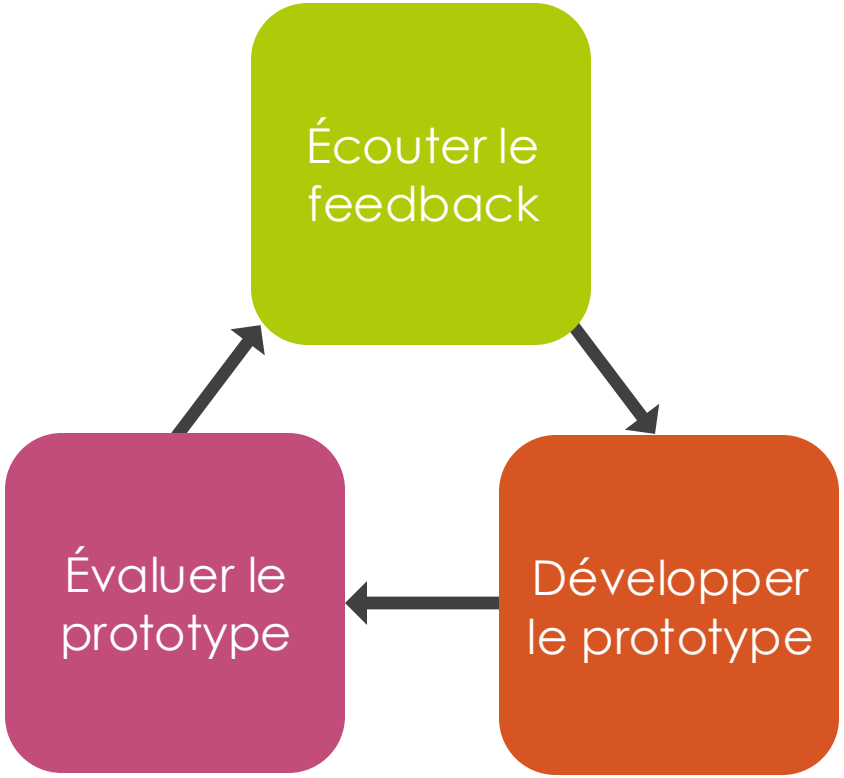
- Prototype affiné selon feedback du client
- Si Prototype satisfaisant → normalisé selon les standards et bonnes pratiques



Exemple de cycle « évolutif » : le Prototypage

Modèle très adapté aux projets de R&D

- Le produit est au centre de l'attention
- Implication active du client / utilisateur
- Le développeur apprend « sur le tas »
- Progrès constant et visible
- **Modèle souvent appliqué aux stages… !**



Inconvénients

- Faible maintenabilité, code peu structuré
- Difficile d'établir un planning : le processus peut ne jamais s'arrêter…

Cycles évolutifs et méthodes Agile

1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Les méthodes Agile

Méthode Agile : cycle de vie qui adopte le « Manifeste Agile »

Origines | Manifeste Agile | Accélération, passage « à l'échelle »

< 2000 | 2001 | > 2001 | > 2005

1991 - RAD
1994 - DSDM
1995 - Scrum
1999 - XP
2000 - ASD


Manifeste pour le développement Agile de logiciels

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.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

2003 - FDD
- Kanban Agile
- BDD
2004 - Crystal Clear

2001 - SoS
2006 - Spotify
- DAD
2011 - SAFe
2013 - LeSS

Constat : cycles « classiques » ne sont plus adaptés aux organisations actuelles, en constante évolution

- Rédigé par 17 experts du logiciel (l'*Agile Alliance*)
- Définit des **Valeurs** et **Principes** communs

Les 4 Valeurs Agile

Individus et interactions	plus que	Processus et outils
Logiciel fonctionnel	plus que	Documentation massive
Collaboration du client	plus que	Négociation de contrats
Réagir au changement	plus que	Suivre le plan

Les 12 principes du Manifeste Agile

1. **Livrer rapidement et régulièrement** des fonctionnalités à **forte valeur ajoutée**
2. **Accueillir positivement le changement** de besoins, même tard dans le projet
3. **Livrer fréquemment un logiciel opérationnel** avec des cycles les plus courts possibles
4. **Utilisateurs et équipe doivent travailler quotidiennement** tout au long du projet
5. **Motiver les individus** : leur fournir soutien et environnement, leur faire confiance
6. **Privilégier le dialogue en face à face** pour transmettre l'information

Manifeste Agile

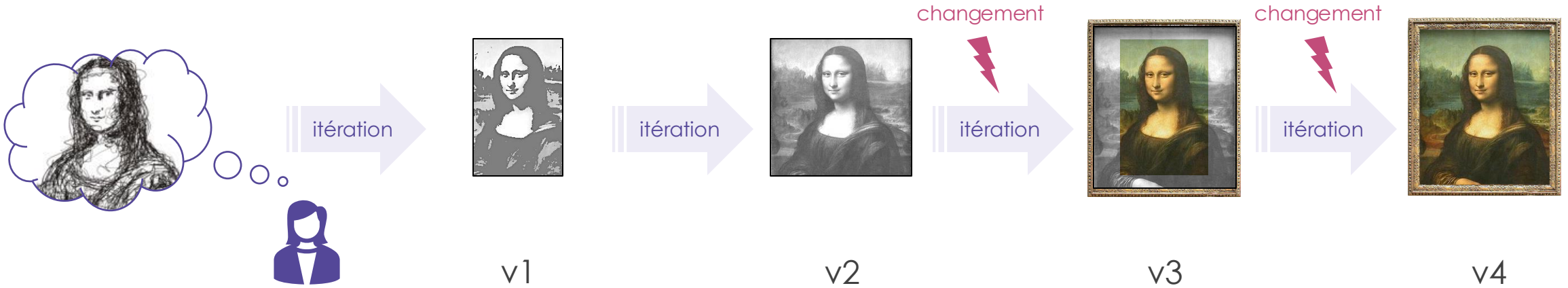
Les 12 principes du Manifeste Agile

Manifeste Agile

7. **Avoir un logiciel opérationnel** est la principale mesure d'avancement projet
8. **Encourager un rythme durable** : le rythme de travail doit être soutenable dans la durée
9. **Être continuellement attentif à l'excellence** technique et la bonne conception
10. **Viser la simplicité** : minimiser la quantité de travail inutile
11. **Une équipe auto-organisée** produit de meilleures réalisations
12. **S'améliorer régulièrement** : réfléchir aux moyens d'être plus efficace, s'adapter en conséquence

Cycle Agile > Cycle « évolutif »

Agile = évolutif (itératif et incrémental) + adaptatif



Gain certain de l'adaptation au changement

Prix éventuel de l'adaptation au changement

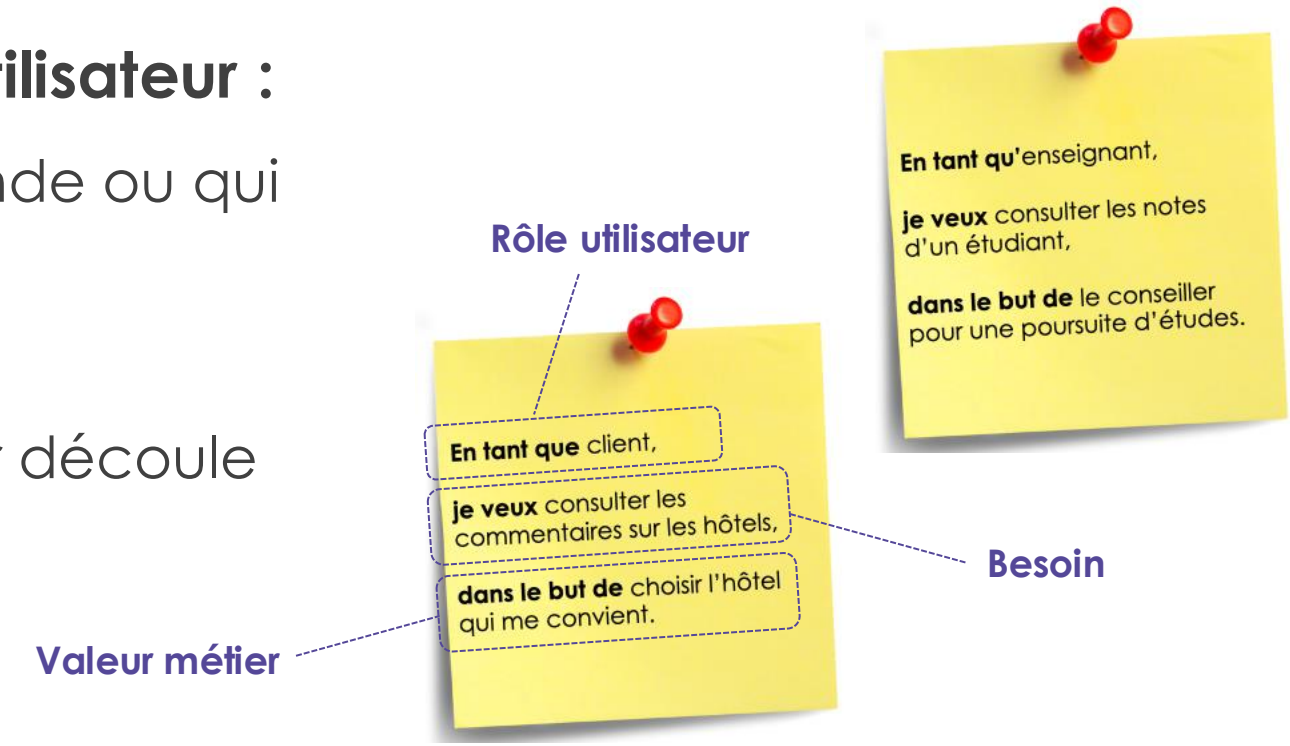
Spécification des Besoins en Agile

User Story (US)

- Description simple et compréhensible d'une **fonctionnalité du système**
- Sorte d'« Exigence » qui **permet l'adaptation au changement**

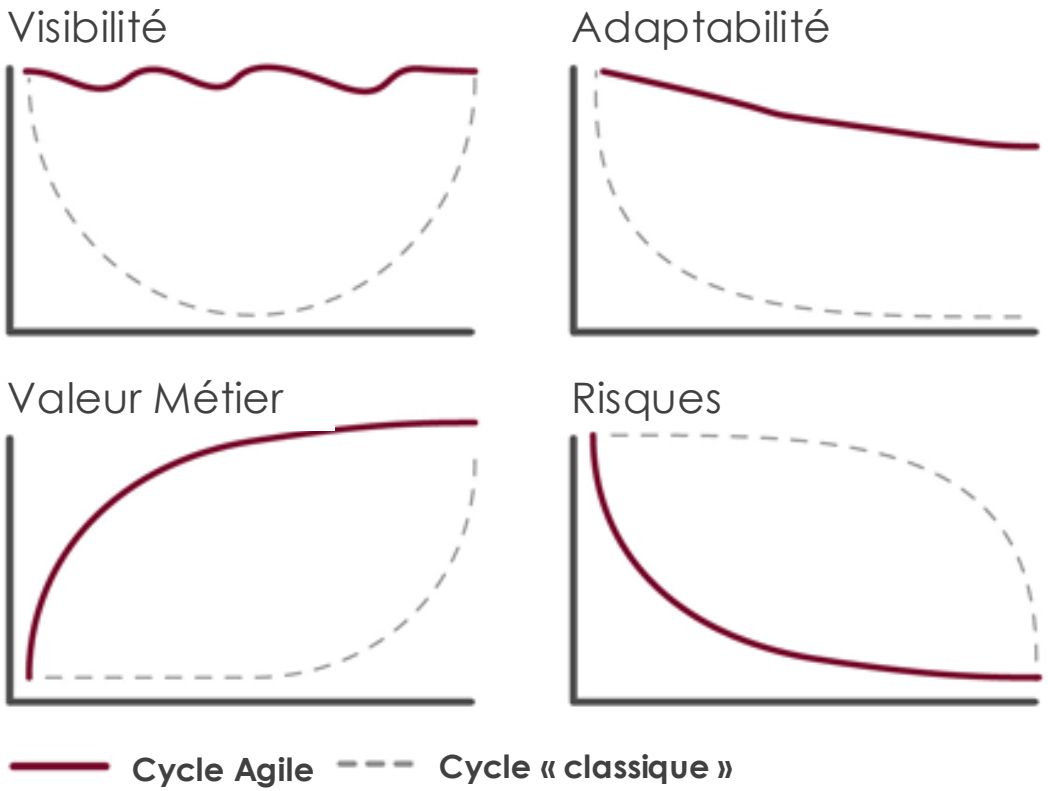
Une US exprime le point de vue utilisateur :

- **Rôle utilisateur** : Qui a fait la demande ou qui bénéfice de la demande ?
- **Besoin** : Quelle est la demande ?
- **Valeur métier** : Quelle valeur métier découle de la réalisation de ce besoin ?



Bénéfices des cycles Agile

« Réconcilier progrès économique et progrès social. » – *Kent BECK*



- Réduction des coûts
- Alignement sur le marché
- Maîtrise du changement
- Durée de vie des logiciels
- Individus responsabilisés
- Communication efficace
- Détection des problèmes

… Mais le succès de ces méthodes repose sur l'implication de chacun !

Cycles évolutifs et méthodes Agile

1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Créée en 1996 par
Ken SCHWABER et Jeff SUTHERLAND

La méthode Scrum

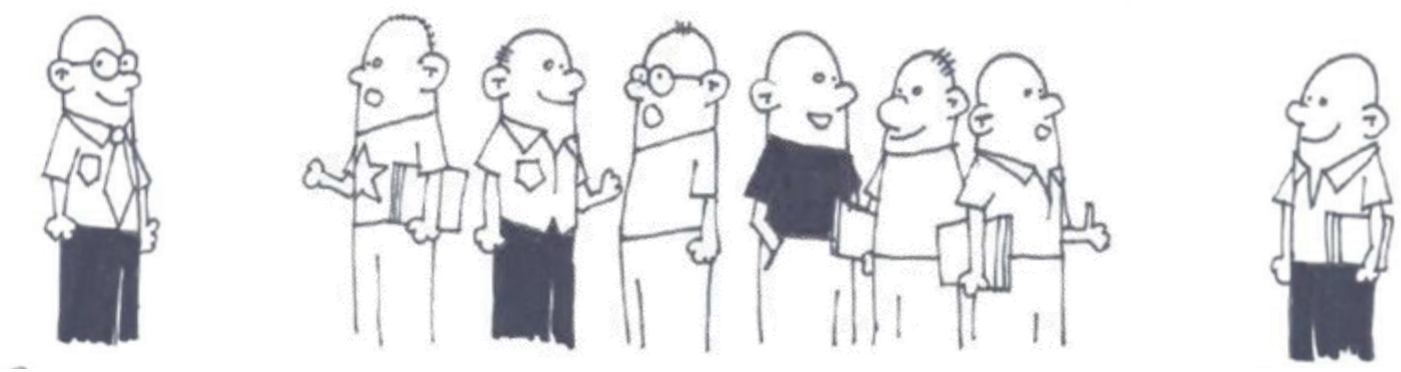
Objectif : améliorer la productivité des équipes

- Éviter l'utilisation de méthodes lourdes
- Produire la plus grande valeur métier dans la durée la plus courte

Fondamentaux de Scrum (= mêlée)

- Itérations d'une durée invariable (Sprint)
- Contenu défini avant chaque itération (Backlog)
- Rôles précis (Product Owner, Scrum Master, Team)
- Réunions structurées (PM, DM, Review, Retrospective)

Les rôles Scrum



Le Product Owner

- Représente clients et utilisateurs
- Définit **la valeur métier** des fonctionnalités
- Travaille avec l'équipe (voire en fait partie)

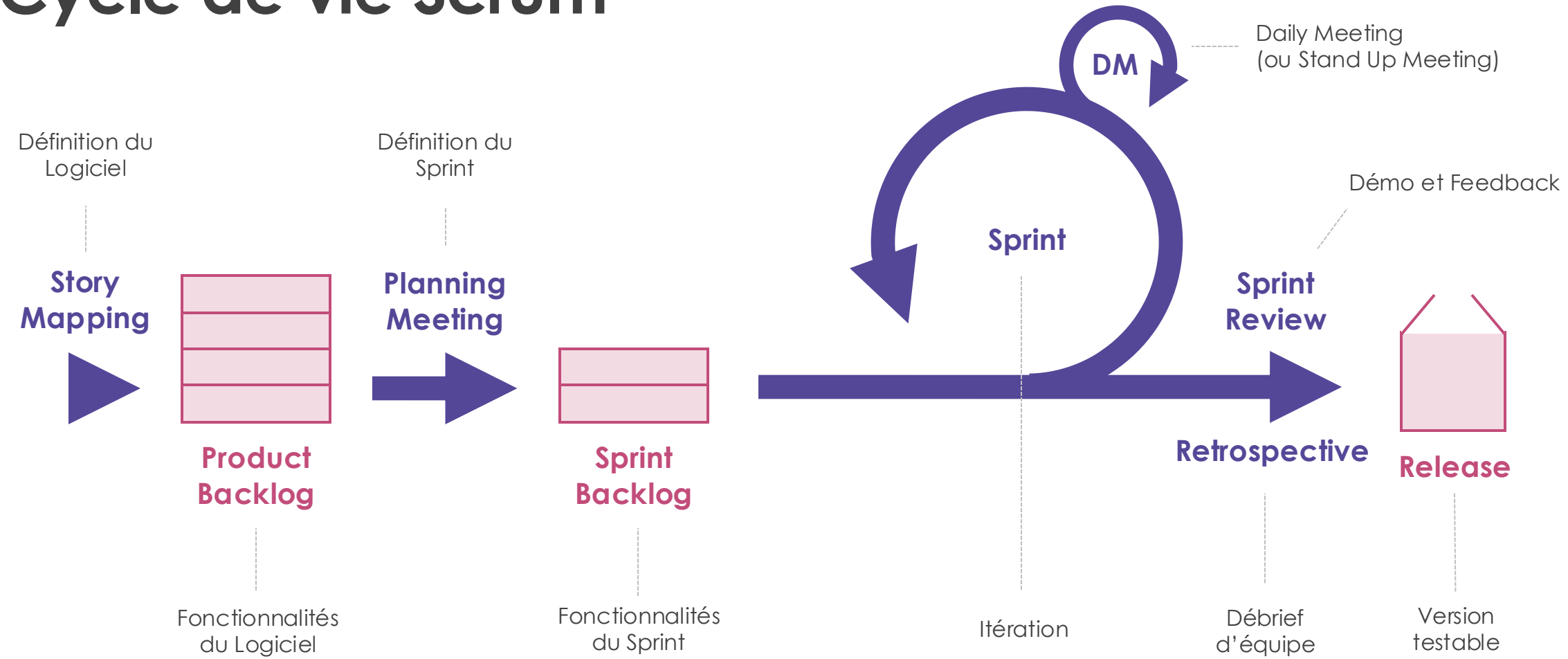
L'Équipe Scrum

- 5 à 10 personnes
- Auto-organisée
- Définit **l'effort** des fonctionnalités
- Développe et teste le logiciel

Le Scrum Master

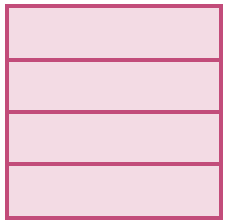
- Anime l'équipe, s'assure qu'elle est fonctionnelle
- Protège des interférences extérieures
- Élimine les obstacles non techniques
- Assure le respect des valeurs de Scrum
- Fait partie de l'équipe (le rôle peut tourner)

Cycle de vie Scrum

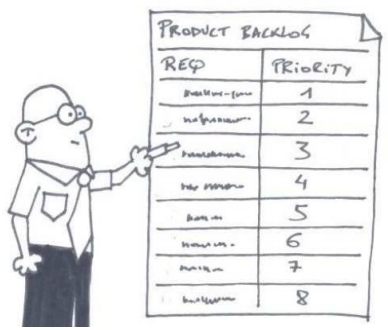


Le Product Backlog

Ensemble des User Stories du Logiciel



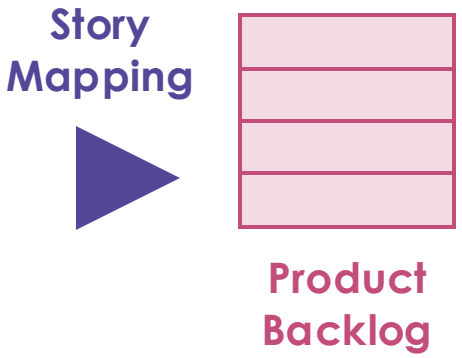
Product Backlog



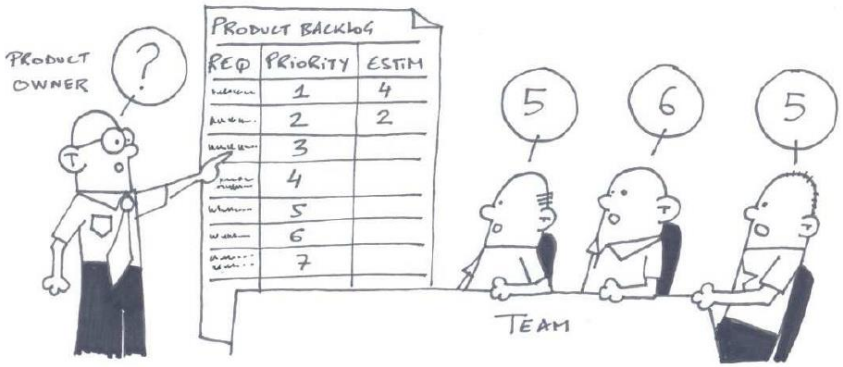
- Regroupées par **Epic** = thématique
- Priorisées par ordre d'**Importance** de réalisation
- Le **Product Owner est responsable** du Product Backlog
- Chaque US comporte :
  - Une **Valeur Métier** = valeur ajoutée pour l'utilisateur final
  - Un **Effort** = complexité (ou temps) pour développer l'US

Le Story Mapping

Séance d'élaboration des User Stories du Product Backlog

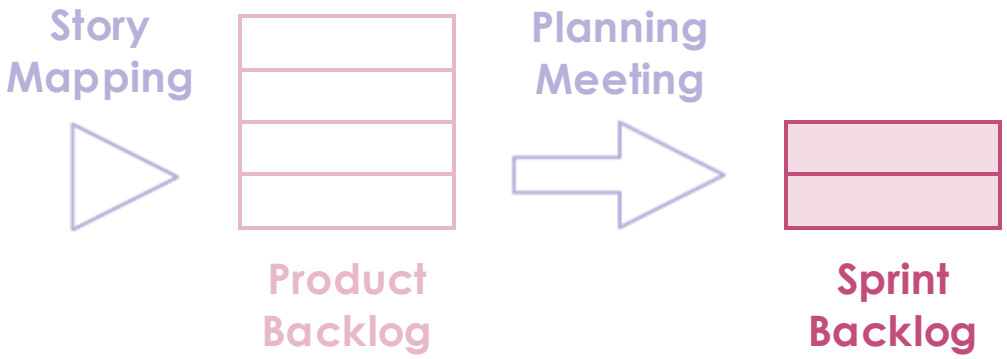


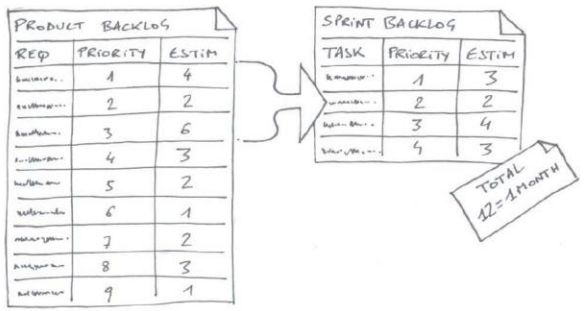
1. Les US sont **créées** et **regroupées** par Epic
2. Le **Product Owner détermine la Valeur Métier** des US
3. L'**Équipe Scrum estime l'Effort** des US (**Planning Poker**)
4. L'**Importance des US est décidée collectivement** au regard de leur Valeur Métier et de l'Effort estimé



Le Sprint Backlog

Sous-ensemble du Product Backlog

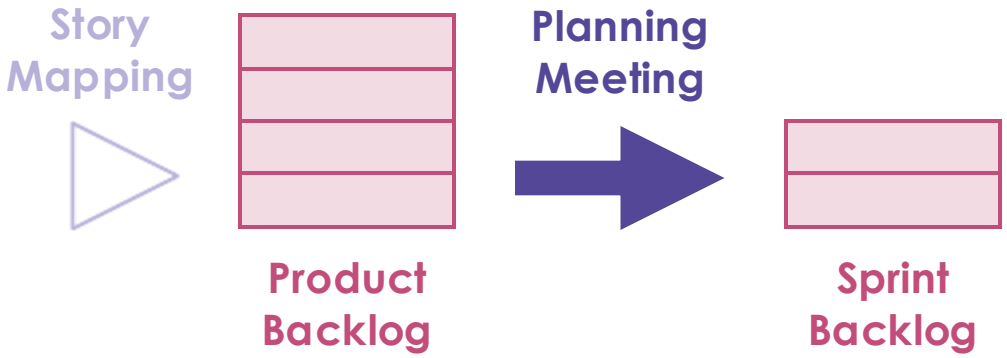




- User Stories du Product Backlog :
  - Que l'équipe **devra réaliser** au prochain Sprint
  - Dont l'Importance est **la plus haute**
- Le Sprint Backlog est construit lors d'une séance de Planning Meeting

Le Planning Meeting

Séance de sélection des US à développer au prochain Sprint



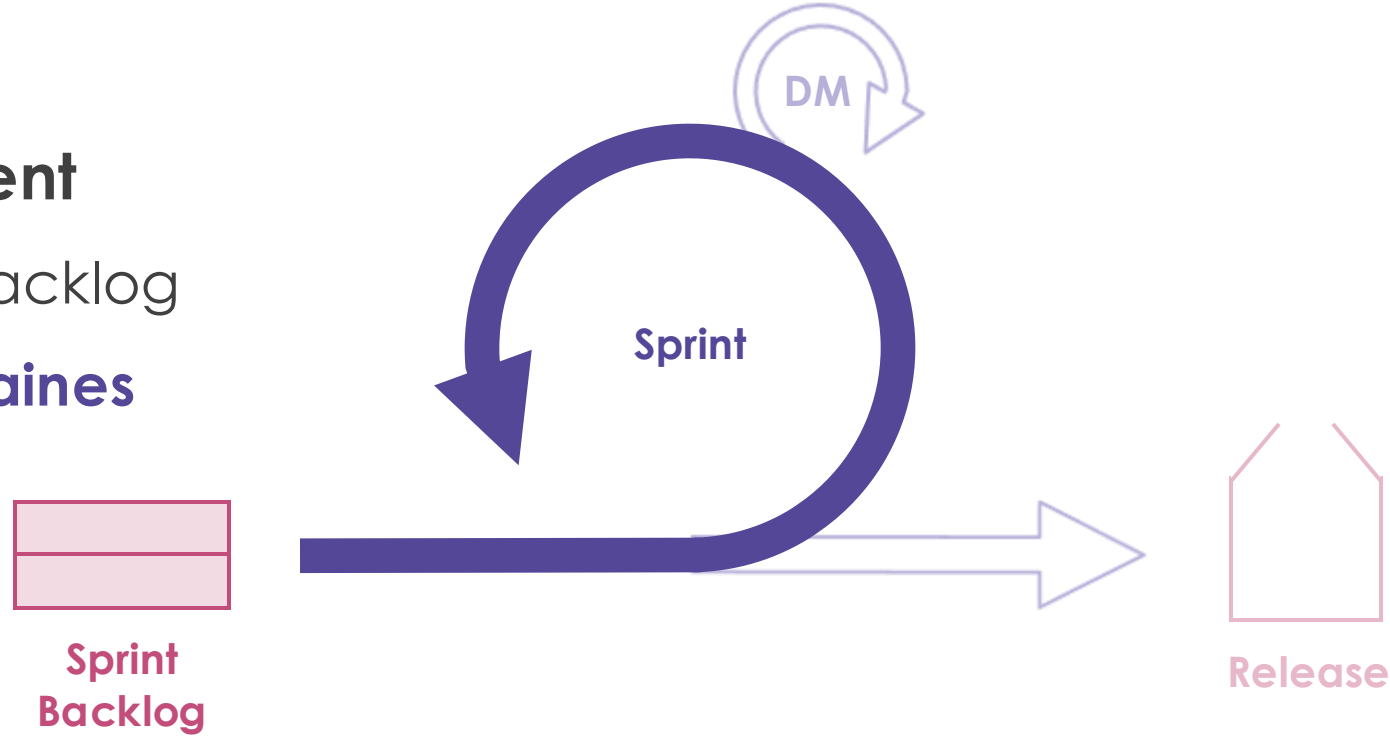
1. Les **Importances des US sont étudiées**
   - Valeur Métier et Effort peuvent avoir changé au cours du Sprint précédent
   - Des US peuvent avoir été ajoutées ou supprimées

2. L'équipe Scrum choisit les US qu'elle **s'engage à finir** sur la durée du Sprint
   - Les US seront **développées par ordre d'Importance**
   - Les US les plus importantes doivent être **les plus détaillées**

Le Sprint

Itération de développement

- Des User Stories du Sprint Backlog
- D'une durée de **2 à 4 semaines**



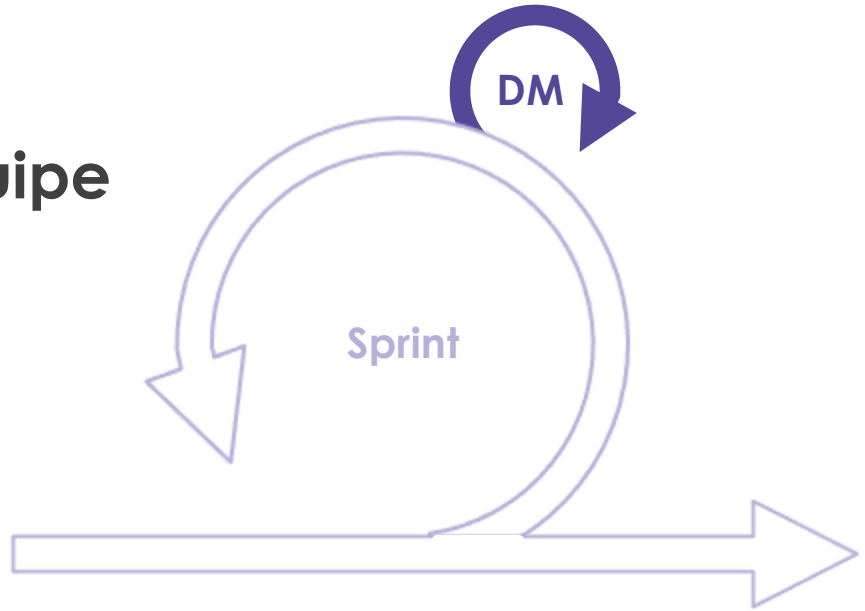
- **US décomposées en tâches**
  - suffisamment petites pour être développées par une seule personne
  - l'effort restant à faire pour chaque tâche **(RAF) est réévalué quotidiennement**
- Le **Sprint Backlog est figé** : aucun ajout ou retrait de User Story pendant le Sprint

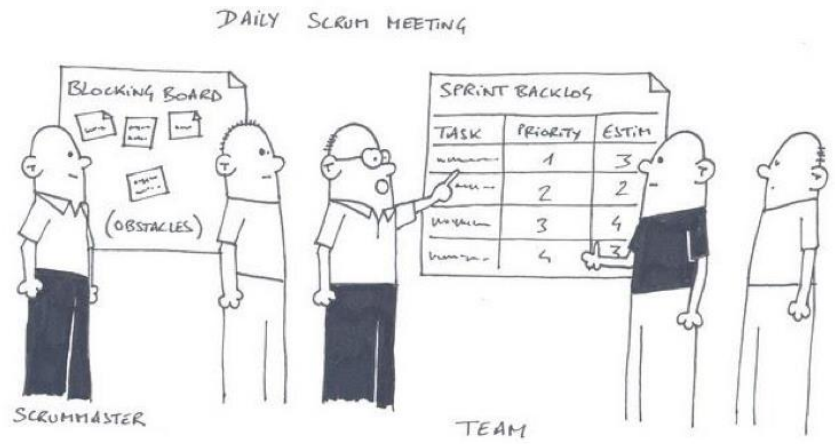
Le Daily Meeting

Point d'échange quotidien de l'équipe

- Chacun résume :
  1. ce qu'il **a fait la veille**
  2. les **problèmes rencontrés**
  3. ce qu'il compte **faire ce jour**

  … et **met à jour le RAF** de ses tâches





- Maximum 2 min / personne
  - Debout
  - Autour du Sprint Backlog
- Identifier les obstacles et problèmes :
  - **ne pas les résoudre** en séance
  - planifier un point dédié si besoin

La Release

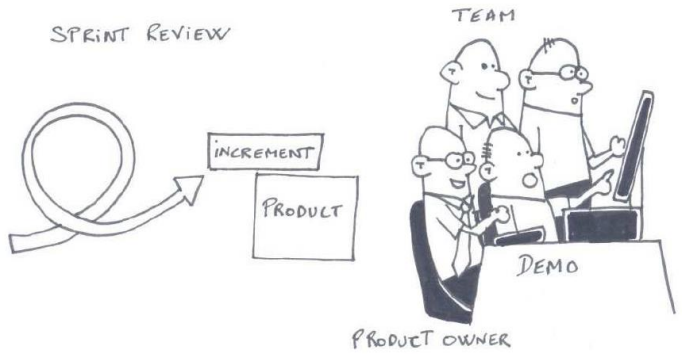
Nouvelle version du Logiciel

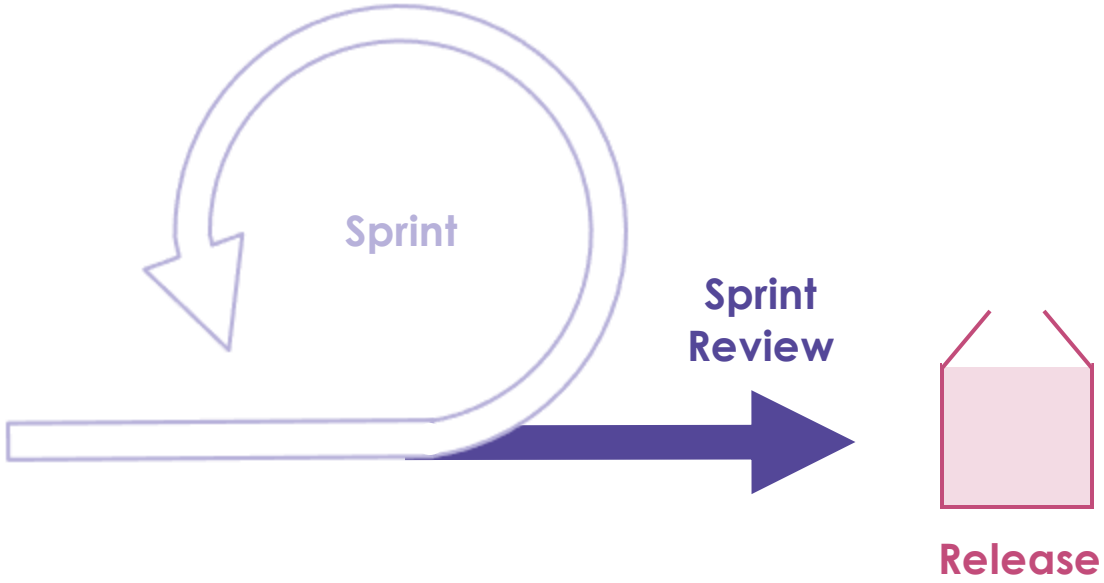
- **Livrée au Product Owner** à la fin du Sprint (en Sprint Review)
- **« Incrémentée » des User Stories** du Sprint Backlog

La Sprint Review

Revue du Sprint

- Démo au PO
  - des **nouvelles fonctionnalités**
  - avant mise en production
- Pour valider
  - la **complétude de la version**
  - l'absence de bug bloquant



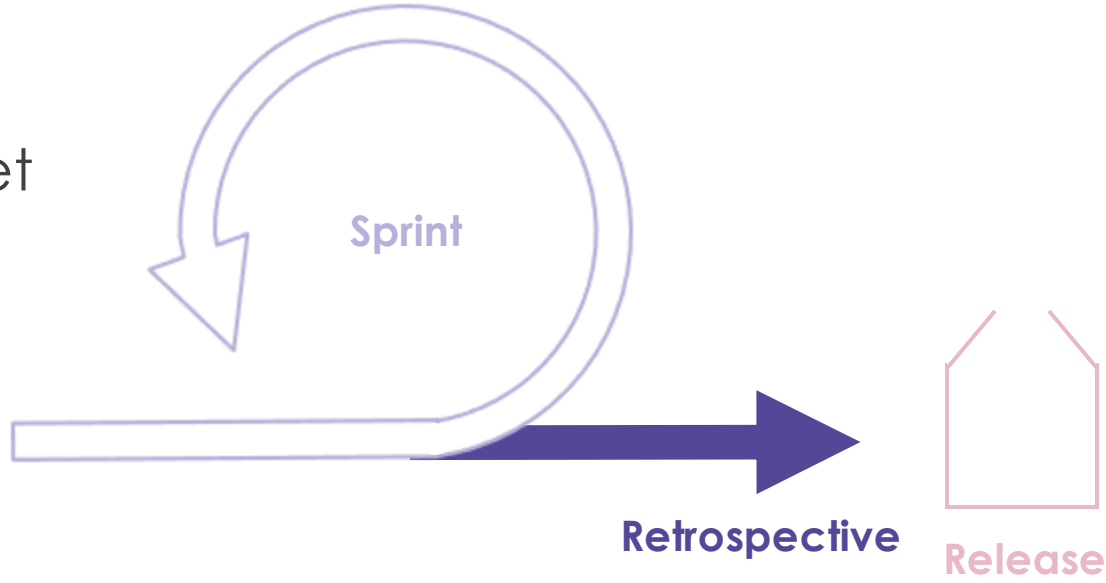


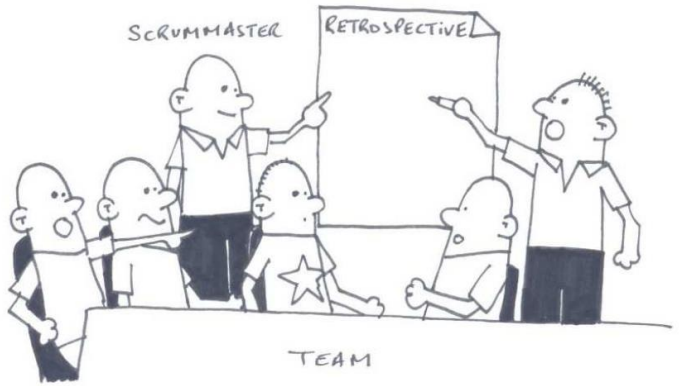
- Pour **mettre à jour le Product Backlog**
  - intégrer les retours utilisateurs (ajout/supp de US)
  - revoir les priorités utilisateur (Valeur Métier)
  - revoir les complexités (Effort)

La Retrospective

À la fin de chaque Sprint

- **Réfléchir au fonctionnement** du projet
  - identifier forces et faiblesses
  - ce qui a et n'a pas marché
  - ce qu'il faut poursuivre
  - ce qu'il faut améliorer





- But : **améliorer le déroulé** du prochain Sprint
  - être « raisonnablement ambitieux » : ne pas chercher à tout améliorer d'un coup
  - choisir 2 ou 3 axes d'amélioration max… mais s'y tenir !

Bénéfices de Scrum

Méthode empirique qui vise l'essentiel

- Détection rapide des anomalies
- Feedback client continu
- Équipe auto-organisée et multi-compétente
- Itérations courtes, résultat rapide et visible

Méthode Orientée projet

- Mise en œuvre simple
- Peut être combiné avec d'autres méthodes
- Adoptée entre autres par les géants du marché (GAFAM)

Limites de Scrum

Méthode sujette à interprétation et variations

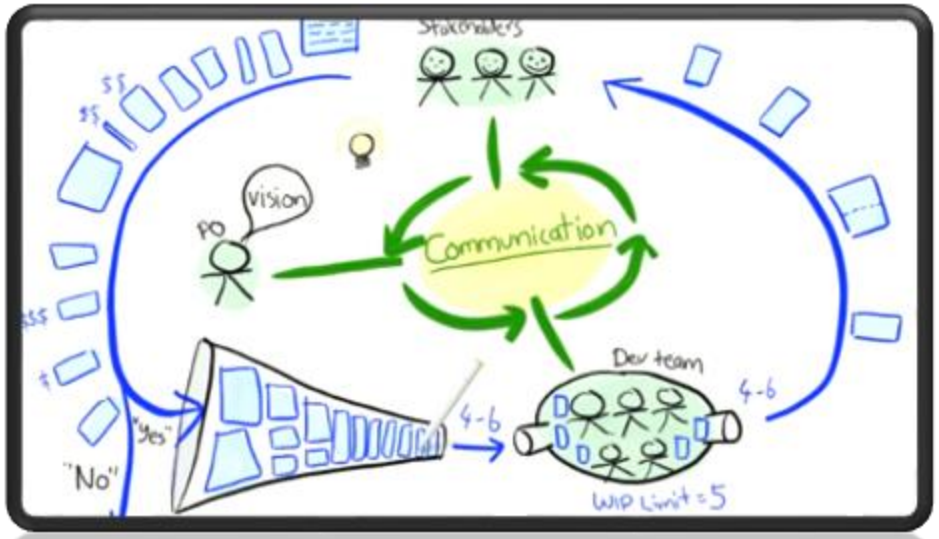
- Sa mise en œuvre diffère selon les organisations
- Nécessite l'implication active de tous les participants

Difficile à mettre en œuvre dans une relation « contractuelle » client/fournisseur

- car souvent basée sur un périmètre fonctionnel fixe
- car souvent contrainte de respecter un budget

Sprint Backlog fixe… et si un User Story est bloqué ?

Vidéo : Gestion d'un Produit Agile en deux mots



Cycles évolutifs et méthodes Agile

1. Limites des cycles de vie « classiques »
2. Principes des cycles de vie « évolutifs »
3. Les méthodes Agile
4. La méthode Agile : Scrum
5. La méthode Agile : XP

Créée en 1995 par
Kent BECK et Ward CUNNINGHAM

La méthode XP (eXtreme Programming)

Le codage est le centre de l'XP

- Moyen léger, efficace, à bas risques et flexible pour le dév. de logiciels
- Destinée à des équipes de moyenne taille avec spécifications incomplètes

Les 4 piliers de l'XP

- Itérations courtes et livraisons fréquentes
- Implication client et War Room
- Pair Programming
- Simplicité et amélioration continue

Itérations courtes et livraisons fréquentes

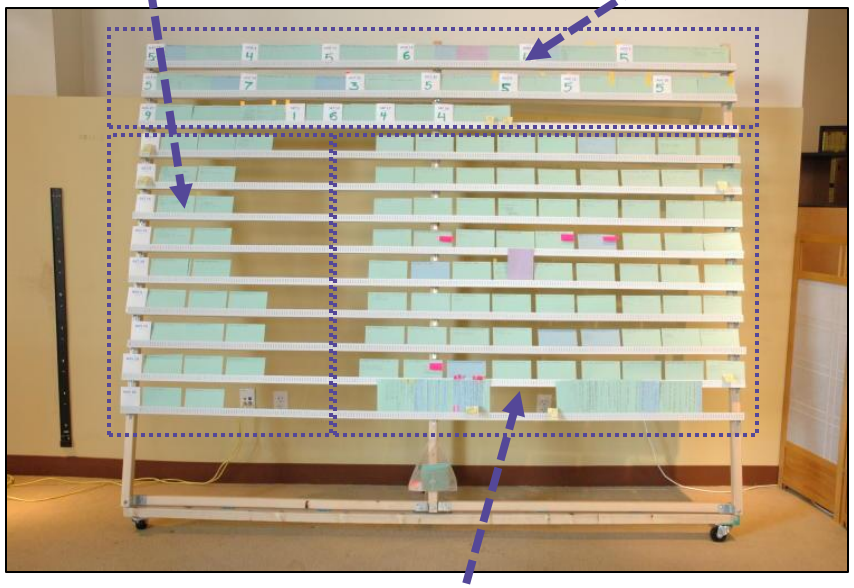
Cycle de vie

- Sélection de « Cards » : scénarios à réaliser
- Définition et répartition des tâches
- Planification, développement et tests
- Fourniture d'un logiciel exécutable et évaluation

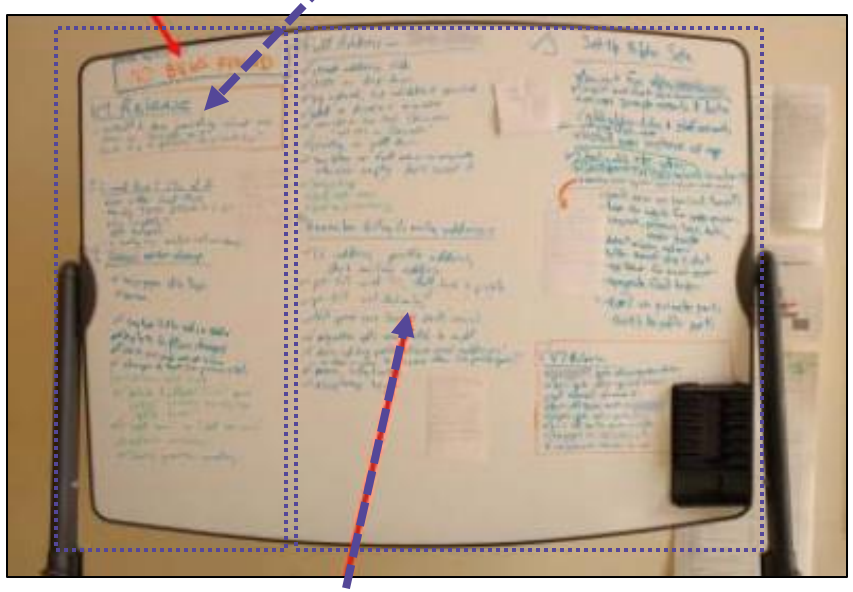
- Livrer un premier système minimaliste
- Le faire évoluer avec des délais très courts

Itérations courtes et livraisons fréquentes

Gestion des « Cards »



Suivi des « Cards » courants



Implication client et War Room

Programmation dans une « War Room »

- De préférence chez le client
- Le client est présent en permanence pour participer, répondre aux questions

« Planning Game »

- Clients et développeurs décident du contenu de chaque livraison en priorisant les spécifications
- Seuls les développeurs sont responsables d'estimer la charge de travail

Pair Programming

Le code est écrit par deux personnes par machine

Le *Driver* écrit le code

Le *Navigator* relit et corrige

Rotation régulière :

- Des rôles : les rôles de *Driver* et *Navigator* s'inversent
- Des binômes : les développeurs changent de binôme

Pair Programming

Favorise la diffusion de la connaissance et la propriété collective

- **Revue permanente** du code : moins coûteuse qu'une revue formelle
- N'importe qui peut changer n'importe quoi, à n'importe quel moment
- Le code est à tout le monde (**Egoless Programming**)

⚠ Impose d'établir des **conventions de codage**

Favorise le refactoring

- **Amélioration continue** du code tout en veillant à le garder fonctionnel
- Le code source **tend vers la simplicité**

Simplicité et amélioration continue

Utilisation des « Cards »

- Expression naturelle et simple des fonctions/scénarios
- Conception simple (mais non simpliste) et partagée entre client et devs

Sont préconisés :

- **Intégration continue** : compilation & test automatiquement à chaque fin de tâche
- **Test Driven Development** (TDD) : tests d'une fonction rédigés avant le codage
  - Les développeurs rédigent les tests unitaires de manière continue
  - Le client rédige les tests d'acceptation des fonctionnalités

Bénéfices de la méthode XP

Responsabilisation et solidarité de l'équipe

- **Implication active** du client
- Forte **réactivité** des développeurs

Gains liés au Pair Programming

- Au moins aussi **productif** que deux programmeurs indépendants !
- Appel aux **meilleures pratiques de développement**

Souplesse « extrême »

Limites de la méthode XP

Cadence « intense »

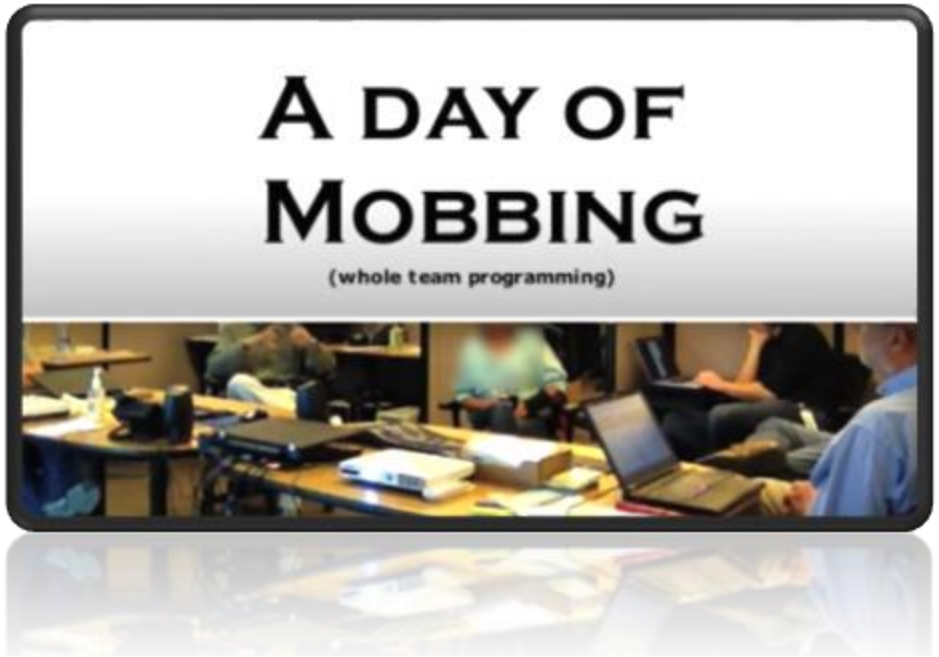
- Stress dû à l'**obligation d'intégration continue et de livraisons fréquentes**
- Demande une certaine maturité des développeurs

Gestion de projet

- La faible documentation peut nuire **en cas de départ des développeurs**
- **Difficulté de budgétiser et planifier** un projet

Pair Programming pas toujours applicable

Vidéo : du Pair Programming au Mob Programming



Université de Rennes

IUT de Lannion

Merci