



Conduite de Projets

Ingénieurs 2^o et 3^o année

II c – Méthodes agiles

1 – Agile Manifesto



Les méthodes agiles

- ❑ Mouvement né dans la foulée de UP. Il en pousse les principes en insistant sur:
 - La livraison rapide du produit ➔ améliorer la compétitivité
 - L'adaptation au changement des exigences du client ➔ coller au besoin réel
 - L'élimination des activités improductives ➔ améliorer la productivité
 - Un esprit collaboratif (au sein de l'équipe et avec le client)
 - Un formalisme plus léger

Agile Manifesto

❏ Déclaration de 17 experts en développement logiciel

(Publiée en Février 2001)

- Principes, méthodes et pratiques à mettre en œuvre pour améliorer la production de logiciels.
- Convictions issues de longues expérimentations.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.



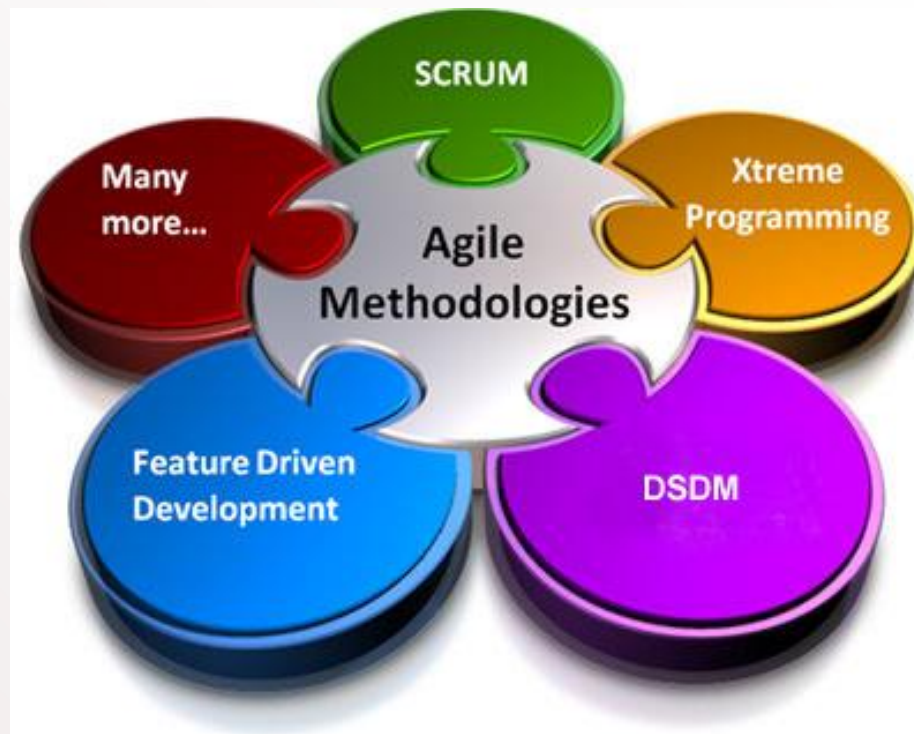
That is, while there is value in the items on the right, we value the items on the left more.

(source : www.agilemanifesto.org)

“

Cf document
« Agile
Manifesto Principles.xlsx »

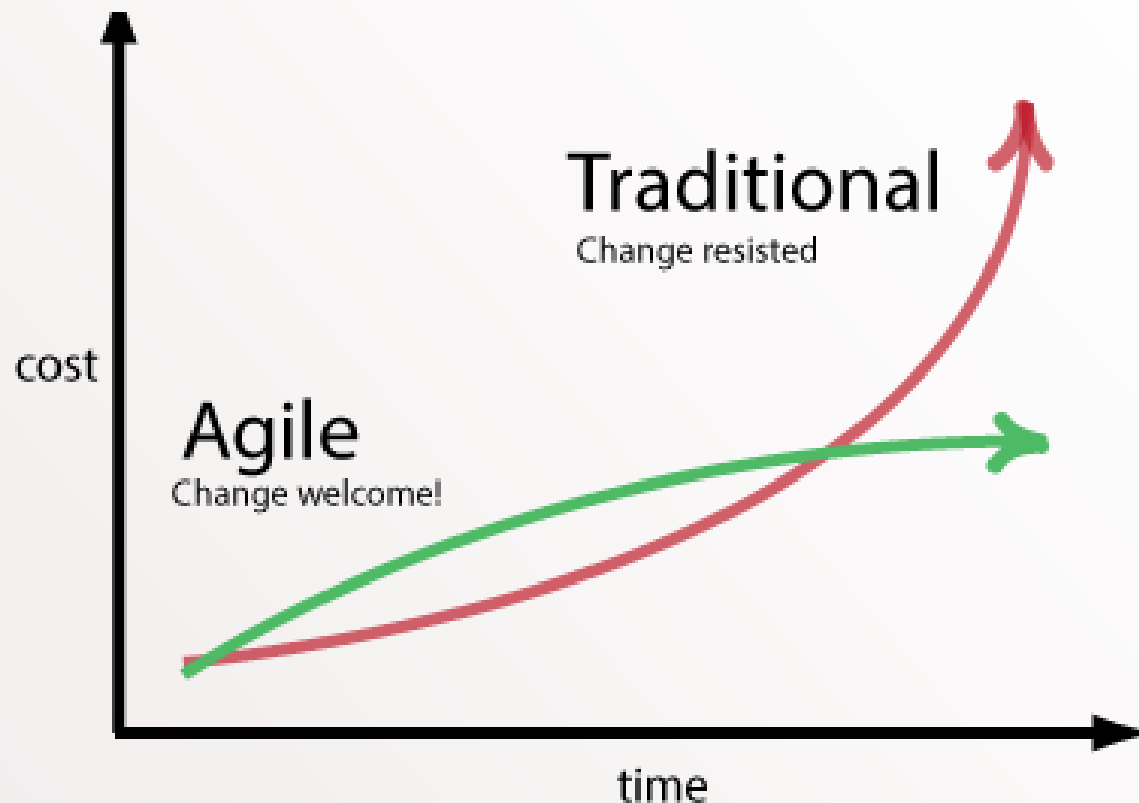
2 – Exemples de méthodes agiles



XP (extreme programming)



- ❑ Idée de base: empêcher un coût exponentiel des changements (1-10-100)




Pratiques XP

- ❑ Petite équipe (< **10 développeurs**), tous sur le même site. Le client est aussi présent.
- ❑ Itération = **1 semaine** → incrément potentiellement releasable.
- ❑ **Feedback ultra rapide** (voire permanent) du client qui voit le système se construire sous ses yeux.
- ❑ Les releases ont lieu environ tous les 3 mois.

[Cf. Schéma « Release et iterations » en scrum](#)

- ❑ Besoins utilisateurs = **User Stories (US)**.
 - Très **petites fonctionnalités**; peuvent être implémentées en une seule itération (incrémentalisme extrême).

 - ❑ **Documentation minimale :**
 - Les US ne sont élaborées qu'oralement (par le client et l'équipe). Leur titre est noté sur un **post-it**.
- 
- Conception **non formalisée**: émerge au fil des améliorations successives du code.
 - Beaucoup de commentaires de code.
-
- ❑ **TDD (Test Driven Development) :**
 - les tests unitaires sont écrits avant le code. Ils sont automatisés au maximum

- ❑ **Pair programming.** Les développeurs:
 - travaillent par paire (revue de code permanente)



- suivent une organisation stricte du code
 - font systématiquement du refactoring.
- ❑ **10' build:** on doit pouvoir faire un build complet du système et le tester en 10 minutes.
- ❑ **Intégration continue:** tous les changements sont intégrés et testés quotidiennement

❏ Information Radiator

- **Environnement de travail informatif et visuel.** Infos affichées en permanence aux murs et remis à jour par tous.



- ✓ les cartes des US
- ✓ le planning
- ✓ l'avancement du projet

Scrum



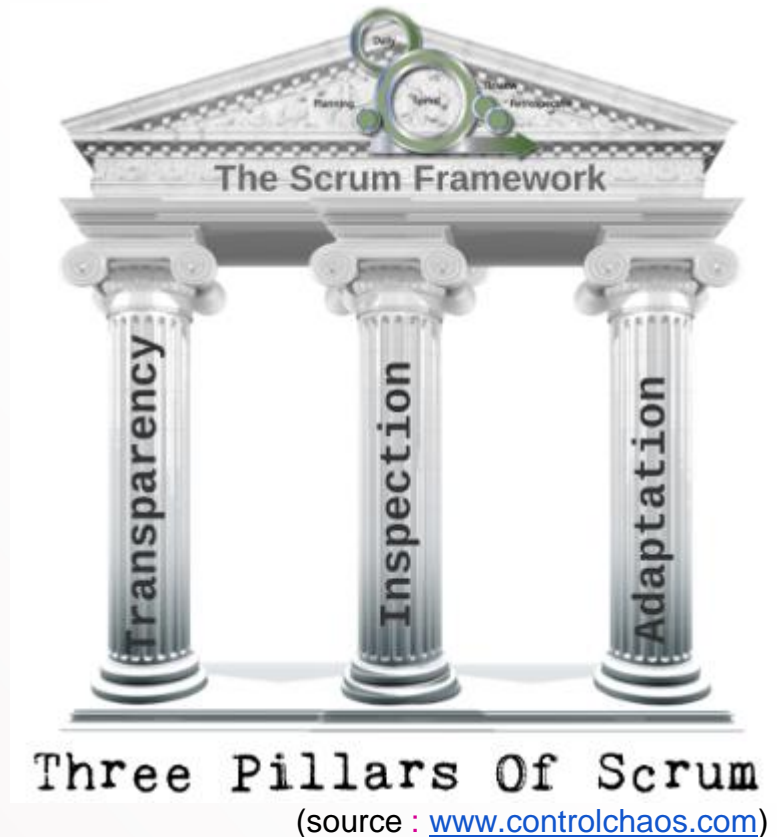
❑ Idée de base: Le développement logiciel est un **processus complexe et imprévisible**.

- Les besoins changent
- La productivité de l'équipe est variable
- La technologie est instable
- Le contexte est en évolution permanente

Approche prédictive non adaptée → Approche empirique avec **adaptations fréquentes**.

❑ Les 3 piliers de Scrum

- **Transparency** = visibilité permanente sur l'avancement du produit
- **Inspection** = évaluation fréquente des travaux en cours, de l'avancement



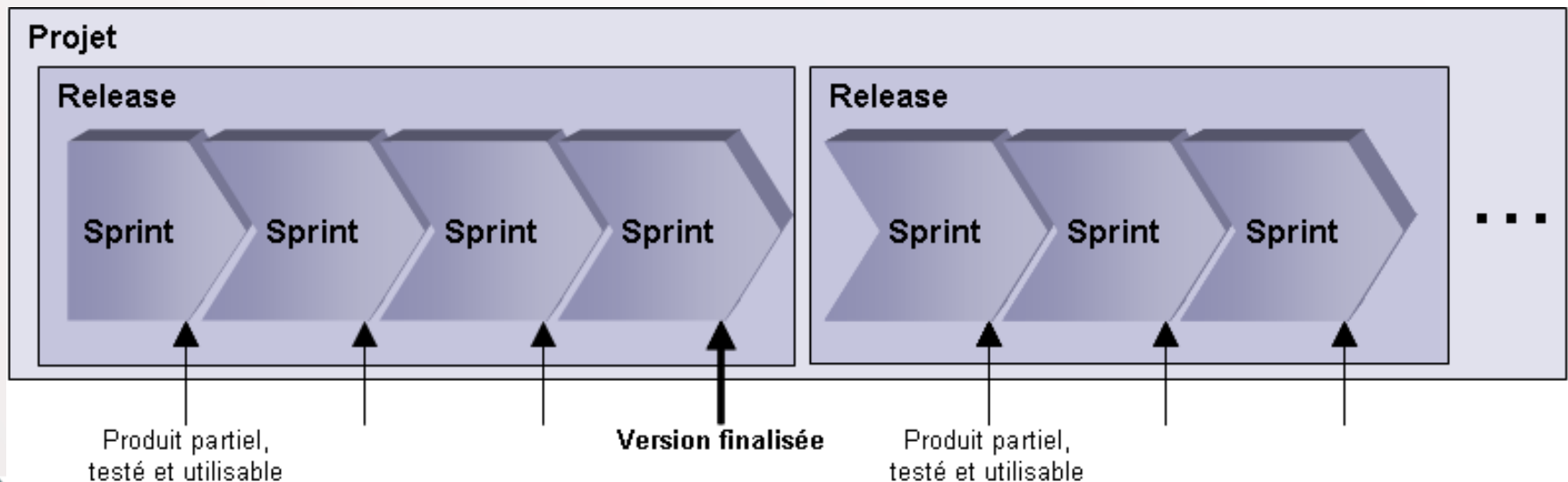
- **Adaptation** = possibilité d'apporter des changements sur la base des évaluations.

➔ Le process doit être constitué d'une série de petites activités, chacune mesurée objectivement.

Pratiques Scrum

- ❑ Approche collaborative, basée sur le travail d'équipe, la communication.
- ❑ Le "Scrum" = **Daily 15' stand-up meeting**. Méthode de communication de base.
- ❑ Le Product **Backlog** = liste de tous les travaux à effectuer (fonctionnalités, bugs, études, etc.)
- ❑ **Time Boxing** : les sprints, les réunions, le projet... doivent impérativement respecter une durée fixée.

- ❑ Itération = "Sprint" de 30 jours. Le résultat de chaque sprint doit être potentiellement releasable.
- ❑ Le périmètre d'un sprint est fixe : une fois que l'équipe s'est engagée sur son contenu, aucune fonctionnalité ne peut être ajoutée.



[Back to XP](#)

- ❑ Petite équipe, tous sur le même site, peu de cloisons fonctionnelles:
 - **Product Owner**: représente les intérêts du client ou des décideurs.
 - **Scrum Master**: aide l'équipe, lève les obstacles, enseigne les règles de Scrum.
 - **Team Member**: multi-compétents (développeurs, testeurs, analystes métier,...).

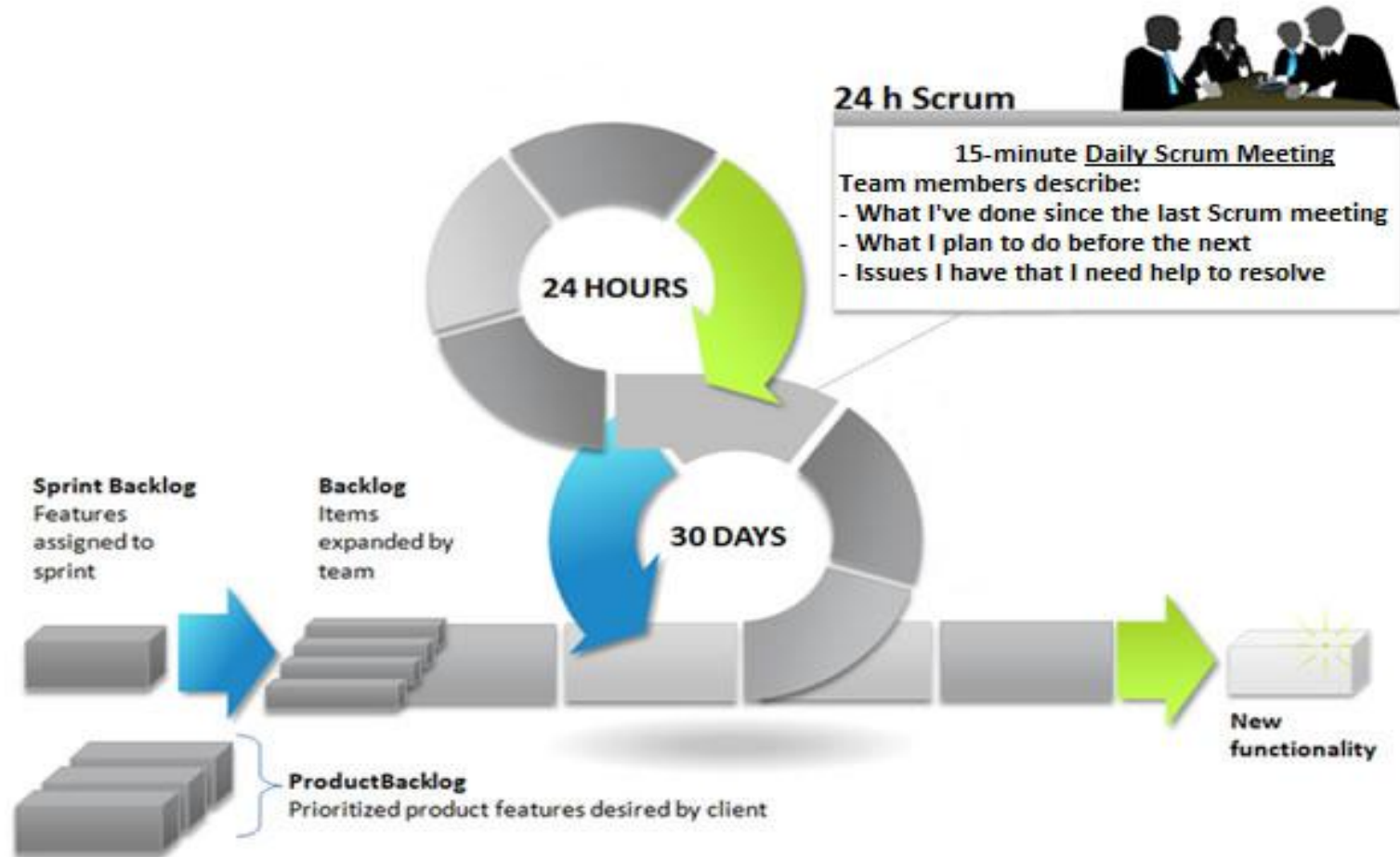


(source : <https://www.braintrustgroup.com/scrum/>)

- ❑ Equipe autogérée :
 - les membres s'organisent, la direction se contente de points de contrôle réguliers.

Modèle simplifié

SCRUM PROCESS

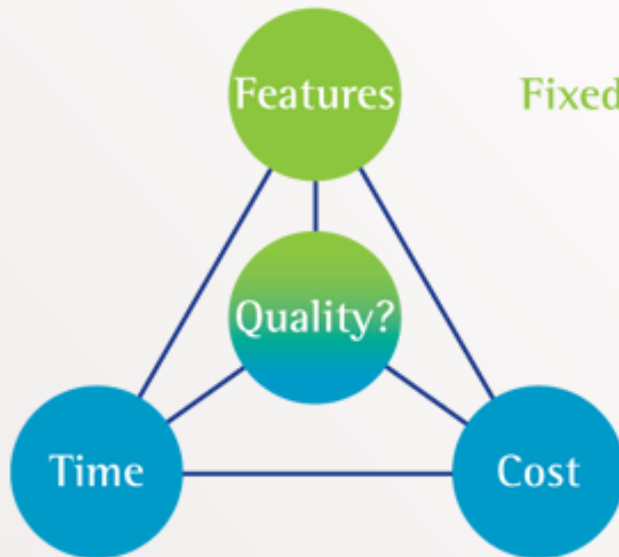


(source : www.controlchaos.com)

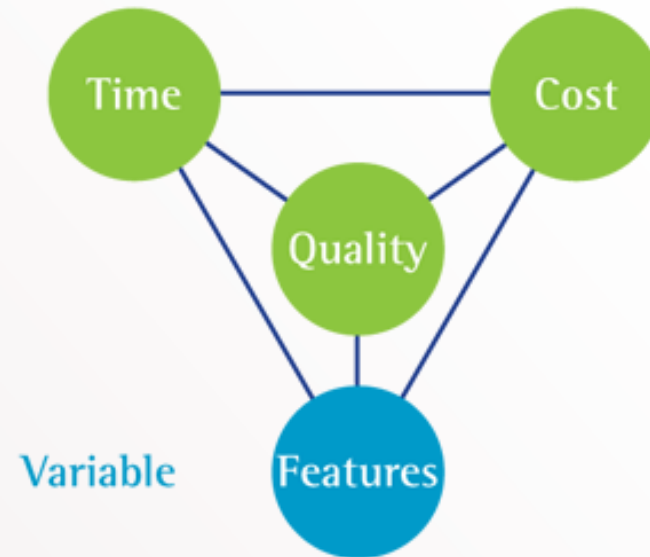
DSDM (Atern)

- Flexibilité du périmètre fonctionnel

Traditional Approach



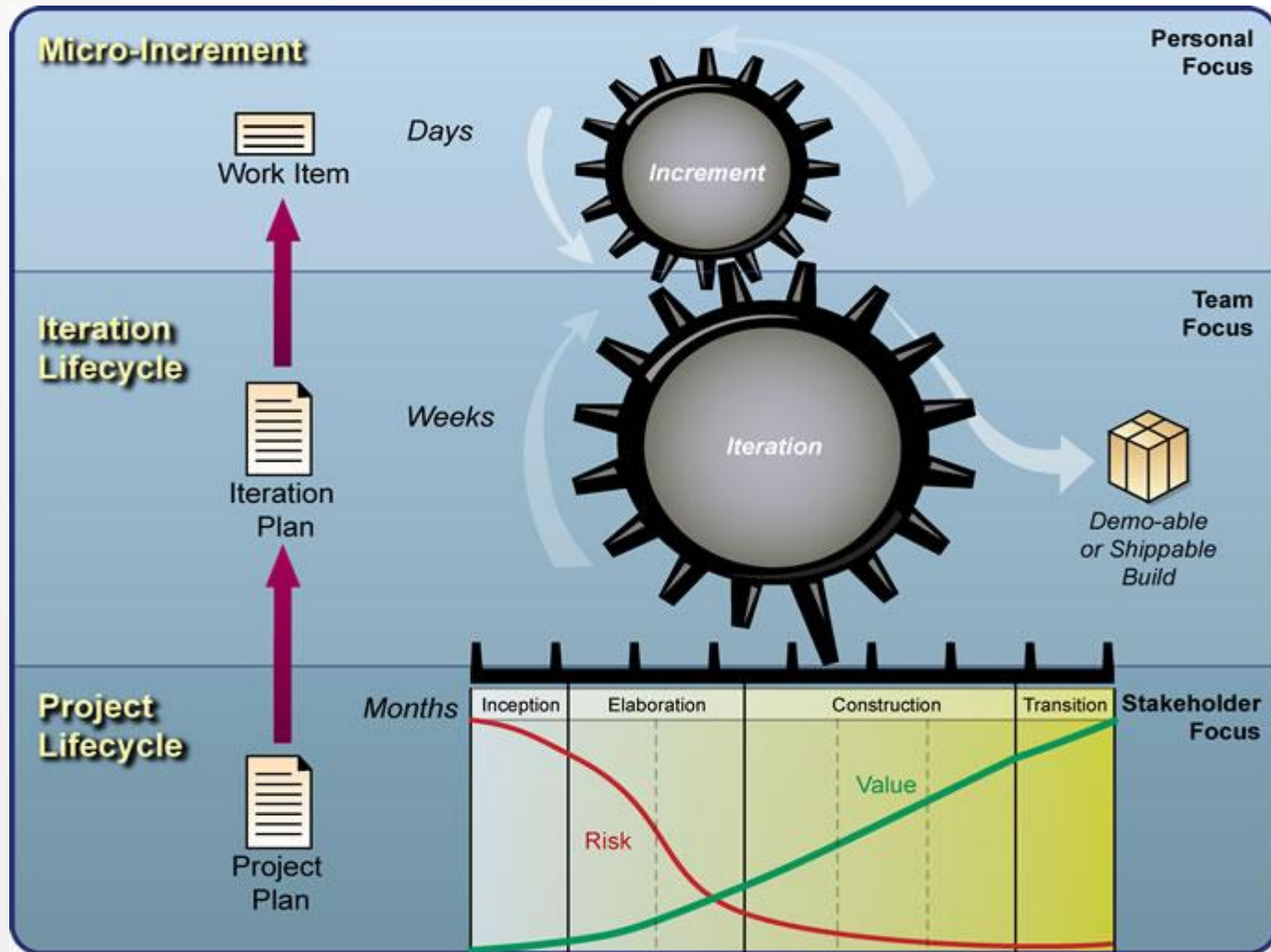
Atern Approach



- ❑ Méthodes traditionnelles: besoins figés (cdc).
 - On adapte les ressources et le délai afin de les réaliser.
 - Retard → La qualité est sacrifiée.

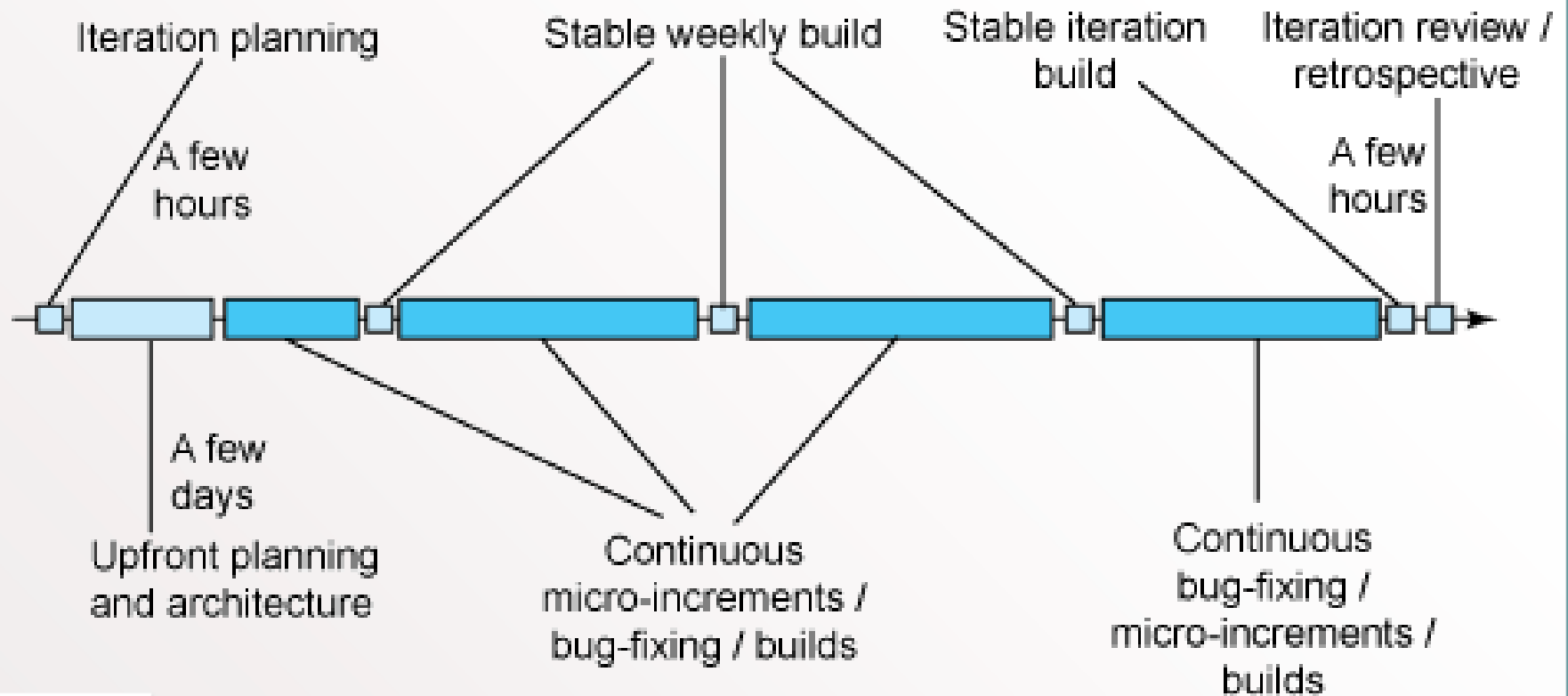
- ❑ DSDM: Les **dates** de livraison sont connues et **non négociables**. Idem pour les ressources.
 - Ce qui sera livré à ces dates est susceptible d'évoluer
 - Les besoins doivent donc être variables, négociables → les prioriser selon la valeur client.
 - La qualité est non négociable

OpenUp: un UP léger et agile



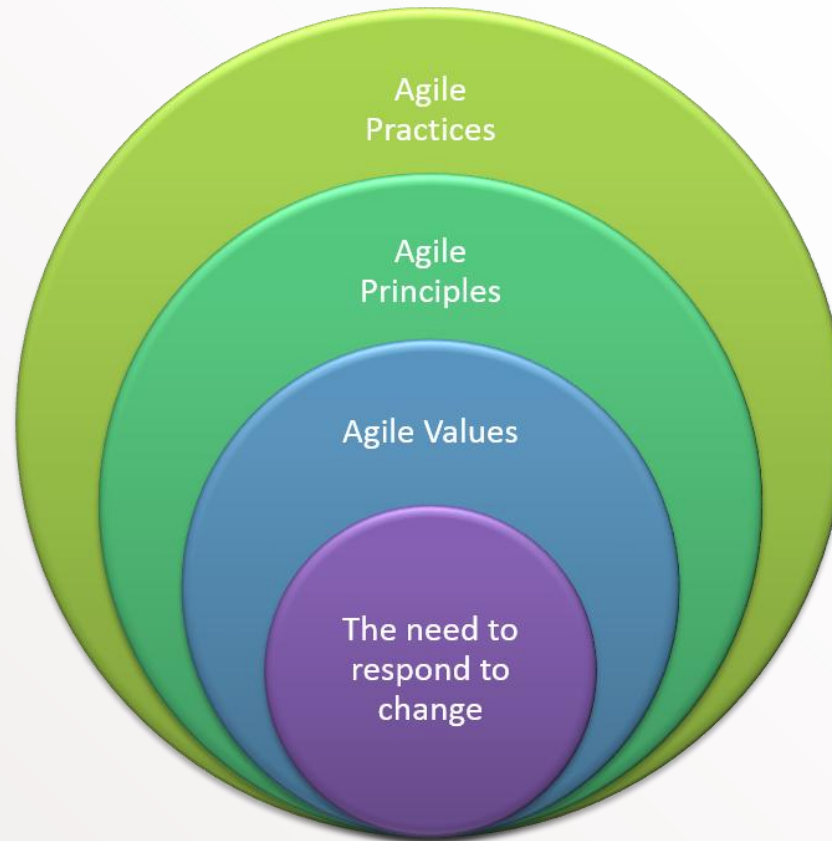
(source : www.ibm.com/developerworks/rational/library/sep07/kroll/)

Contenu d'une itération



(source : www.ibm.com/developerworks/rational/library/sep07/kroll/)

3 – Pratiques agiles



“

Toutes les méthodes
agiles diffèrent mais
elles partagent des
pratiques
fondamentales.

Vision et Backlog

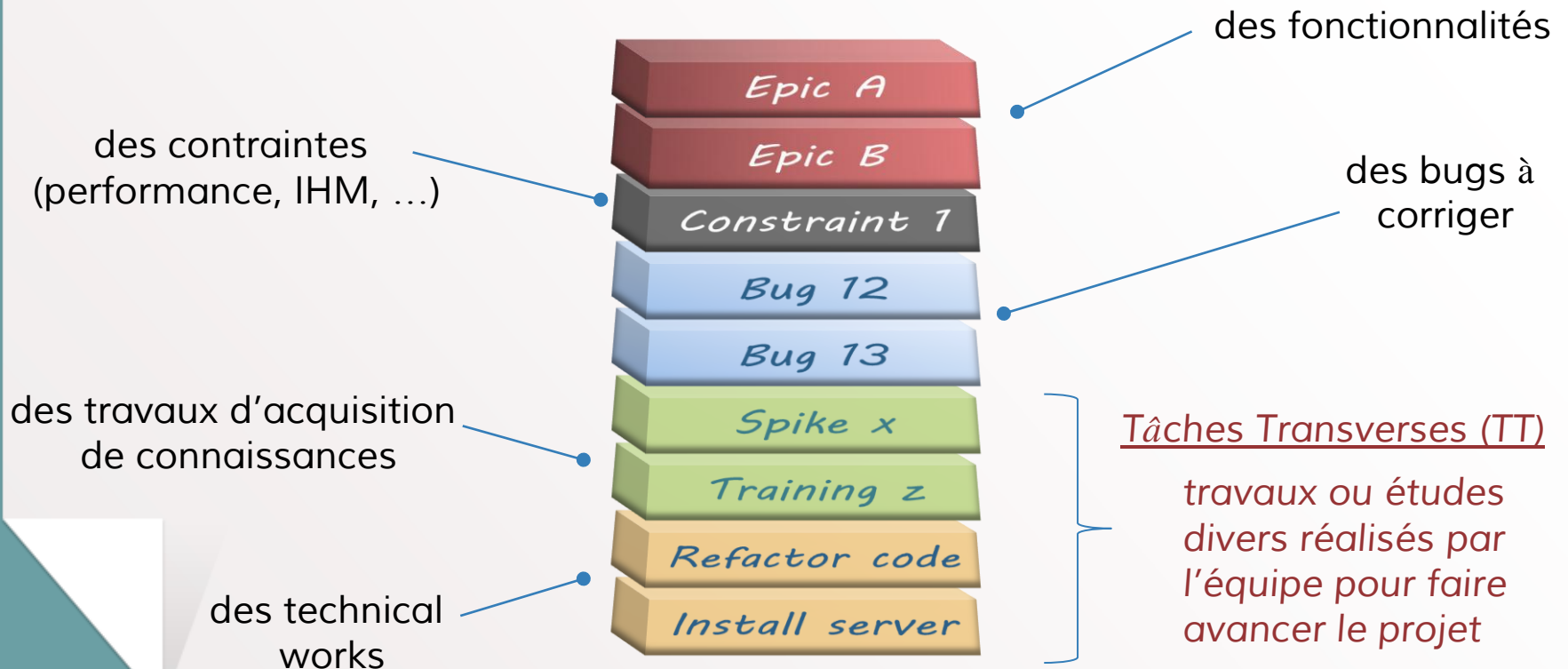
□ Vision Document

- équivalent du cahier de charges. Optionnel mais utilisé par certains agilistes.
- donne une vision d'ensemble succincte du système à développer: objectifs, besoins, contraintes.
 - ➔ vision partagée par le client, l'équipe, les décideurs.
- base pour de futures spécifications plus détaillées

Voir chapitre IV a – Vision

❑ Backlog

- Liste de tous les travaux (« work items ») à effectuer au cours du projet.
- Un backlog contient:

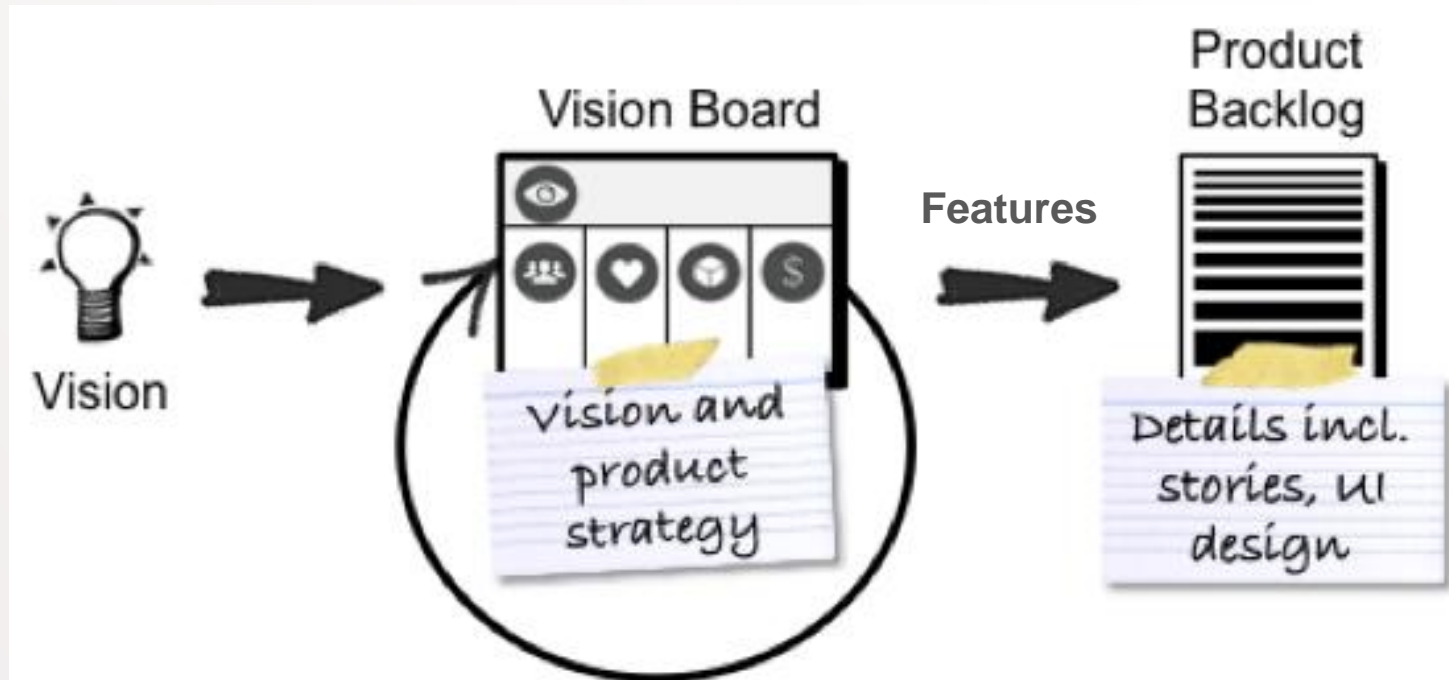


Exemple de Backlog

▼	4.0	2013-06-17	2014-09-12	56 ▲
5032	Bug: Journal Diff formatter renders HTML links regardless of :no_html		in develo...	
7164	Epic: New work packages toolbar		closed	
7184	Epic: Authentication against OpenID connect provider (e.g. google, hero...		closed	
7892	Bug: Spent Time not localized		closed	
1009	Bug: Wrong german translation when accessing a non-existent work pa...		closed	
1502	Bug: Multiple escaping on user select2 field in project members tab		closed	
1865	Bug: Journal creates Symbols from attachment ids		closed	
2029	Bug: [News] Add news: Preview button does not work		closed	
7863	User Story: Encapsulate ActiveSupport::Notifications		closed	
4928	Bug: [Search] Journal note hits not shown		closed	15
5002	Bug: Number of reported work packages in personal activity differs fro...		closed	1
5743	Bug: [security] Potential data leak in "Invalid form authenticity token" er...		closed	1
6309	User Story: Remove API v1		closed	

❑ Vision → Backlog

- Les fonctionnalités et exigences initialement placées dans le backlog proviennent du vision document.



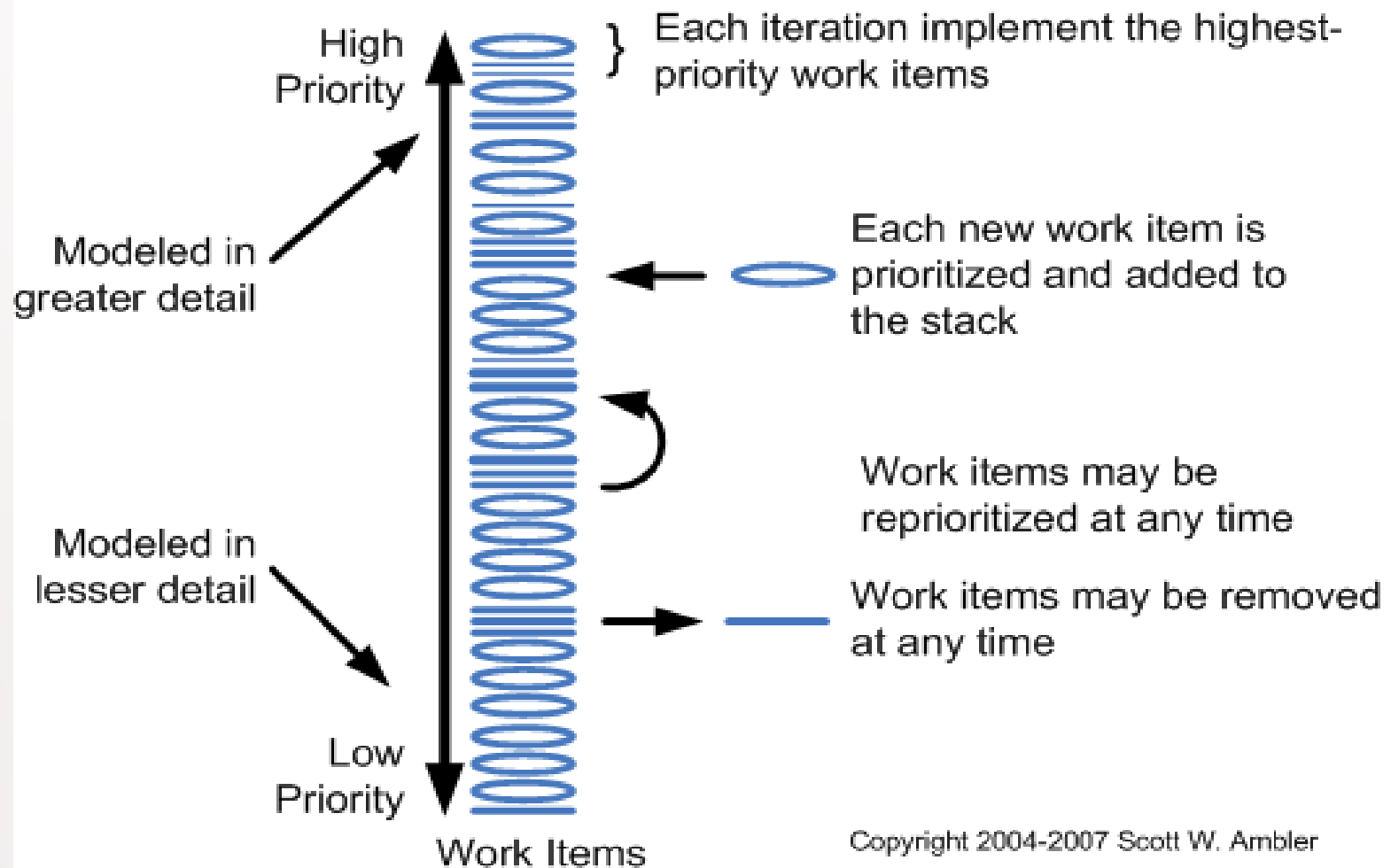
(d'après Mike Cohn: www.mountaingoatsoftware.com/)

❏ Backlog et Planification

- Pour planifier le projet le product owner priorise les work items du backlog:

- ✓ haute priorité: traités à court terme ➔ maximise la valeur client du produit.
- ✓ autres items: « en réserve »; ils seront éventuellement traités plus tard.
- ✓ à tout moment: reprioriser, ajouter ou enlever un item.

TT: sans valeur client directe mais indispensables ➔ Le PO accepte, sur conseil de l'équipe, de les prioriser.



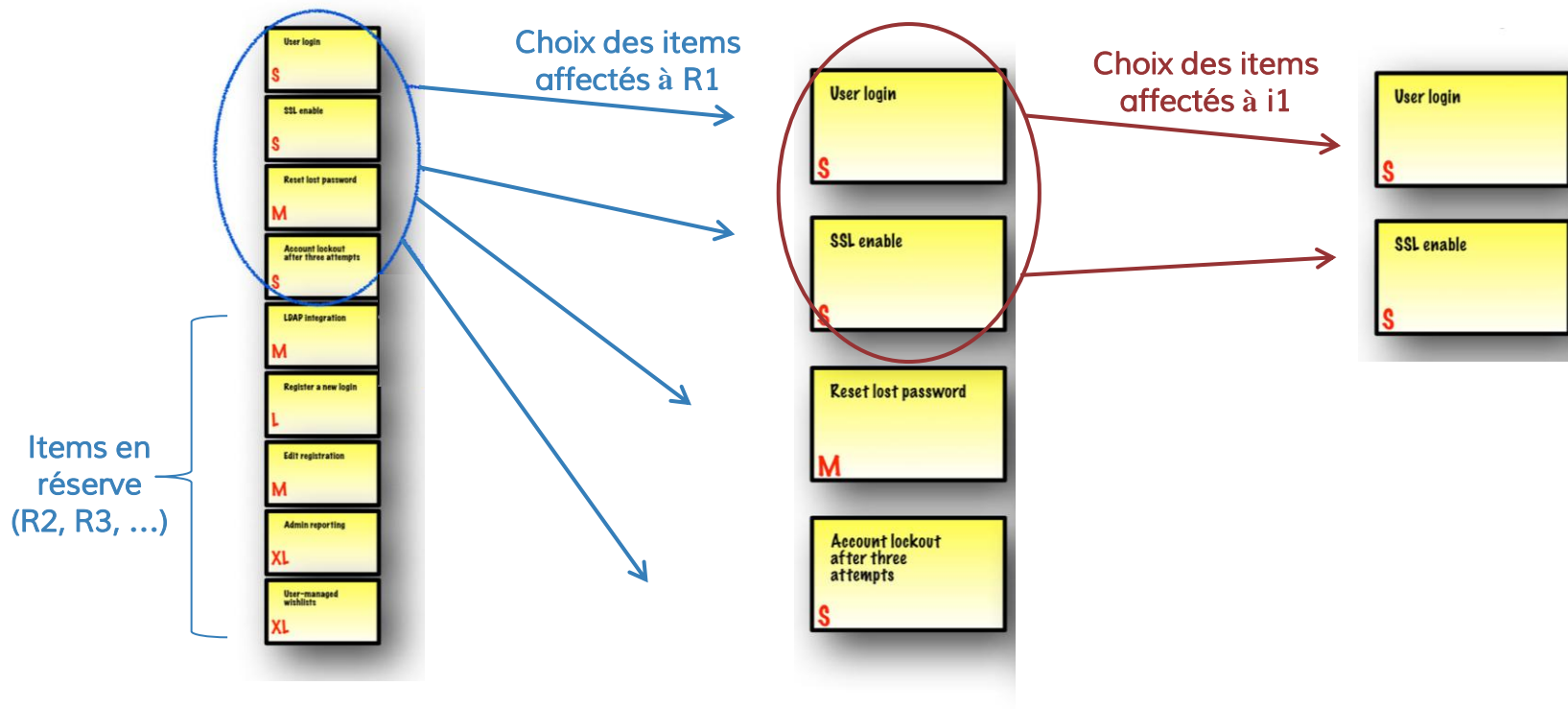
priorisation des items du Backlog

3 types de backlog

Product Backlog
(tous work items du projet)

Release Backlog
(work items réalisés en R1)

Iteration Backlog
(work items réalisés en i1)




NB: modèle simplifié. Ne tient pas compte du split Epics → US.

User Stories

❑ Mode de représentation des besoins fonctionnels en agile

- Elles sont formulées en une ou deux phrases dans le langage de l'utilisateur sous la forme:

« En tant que [utilisateur] je peux [actions de l'utilisateur sur le système] afin de [but utilisateur] ».



As a <role>
I want <goal>
So that <benefit>

Acceptance criteria:

...

Exemples

- « En tant qu'étudiant je peux choisir mes cours en ligne afin de définir mon planning annuel ».
- « En tant que conseiller commercial je peux consulter la liste de toutes les commandes en cours de mon client afin de l'informer sur ses délais de livraison ».
- « En tant qu'opérateur HelpDesk je peux rechercher mon client par son nom ou prénom afin que son temps d'attente soit le plus court possible ».

Support: US Card

- La US est manuscrite (carte bristol ou post-it)

Front of Card

173

As a student I want to purchase
a parking pass so that I can
drive to school

Priority: ~~High~~ Should
Estimate: 4

Back of Card

Confirmations:

~~The student must pay the correct amount~~
One pass for one month is issued at a time
The student will not receive a pass if the payment
isn't sufficient

The person buying the pass must be a currently
enrolled student.

The student may only buy one pass per month.

Copyright 2005-2009 Scott W. Ambler

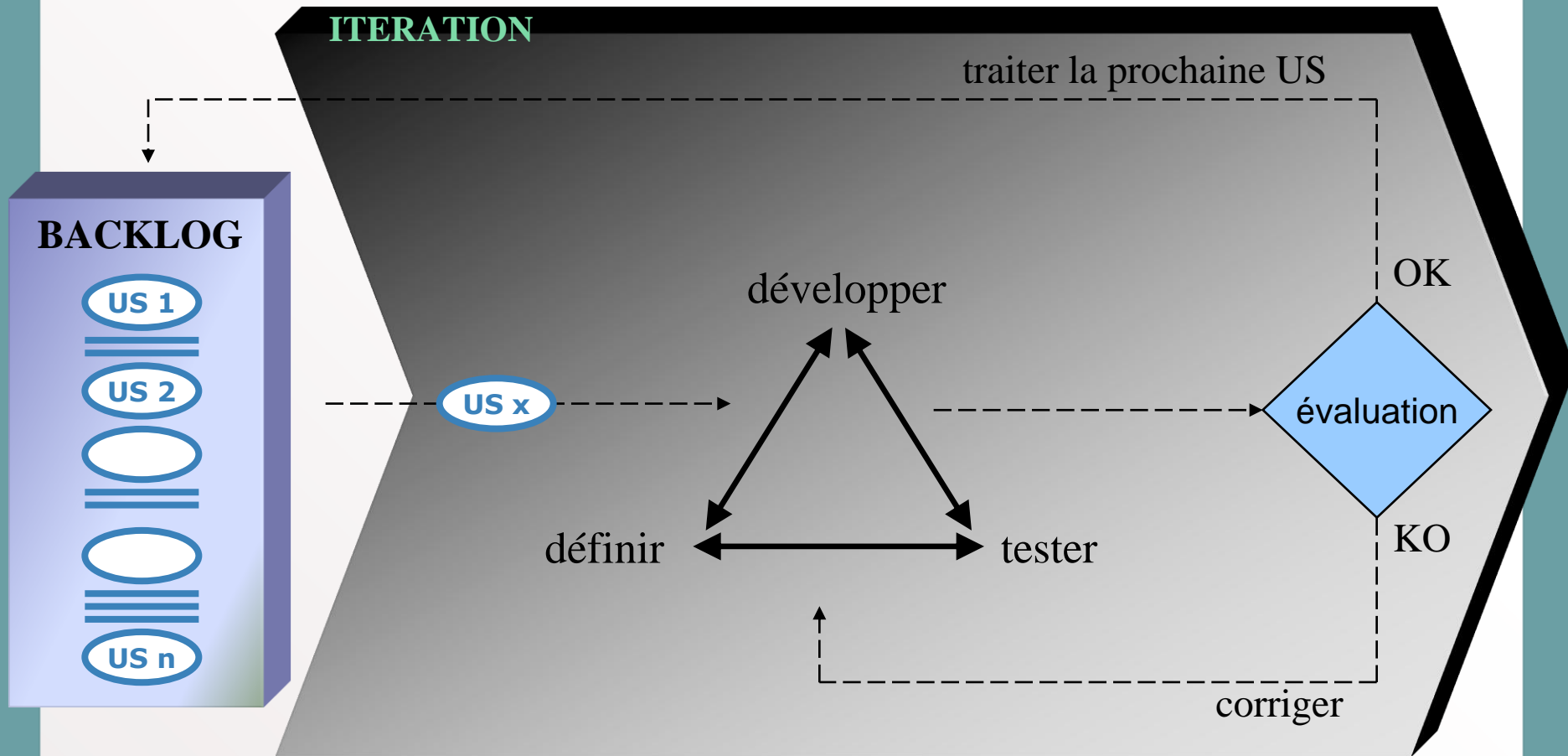
- Les cas de tests sont rédigés au dos

Attributs des US : « INVEST »

- **Independent** : les US ont un minimum de dépendances les unes avec les autres.
- **Negotiable** : le contenu de la US est flexible et peut être négocié avec le client.
- **Valuable** : la US apporte de la valeur au client.
- **Estimable** : la charge d'une US doit pouvoir être estimée avec une bonne précision.
- **Small** : Cf section suivante (granularité des US).
- **Testable** : un test objectif doit permettre de vérifier si la US est acceptable par le client

Granularité des US

- ❑ Une US doit pouvoir être entièrement **traitée en une seule itération.**
 - Les US doivent donc être **petites** et représenter quelques jours de travail.
 - Avantages:
 - ✓ la charge d'une petite US est plus facile à estimer
 - ✓ une petite US sera plus rapidement terminée → mesurer un **progrès objectif au cours de l'itération**



Une US est entièrement traitée en une seule itération

Split des US

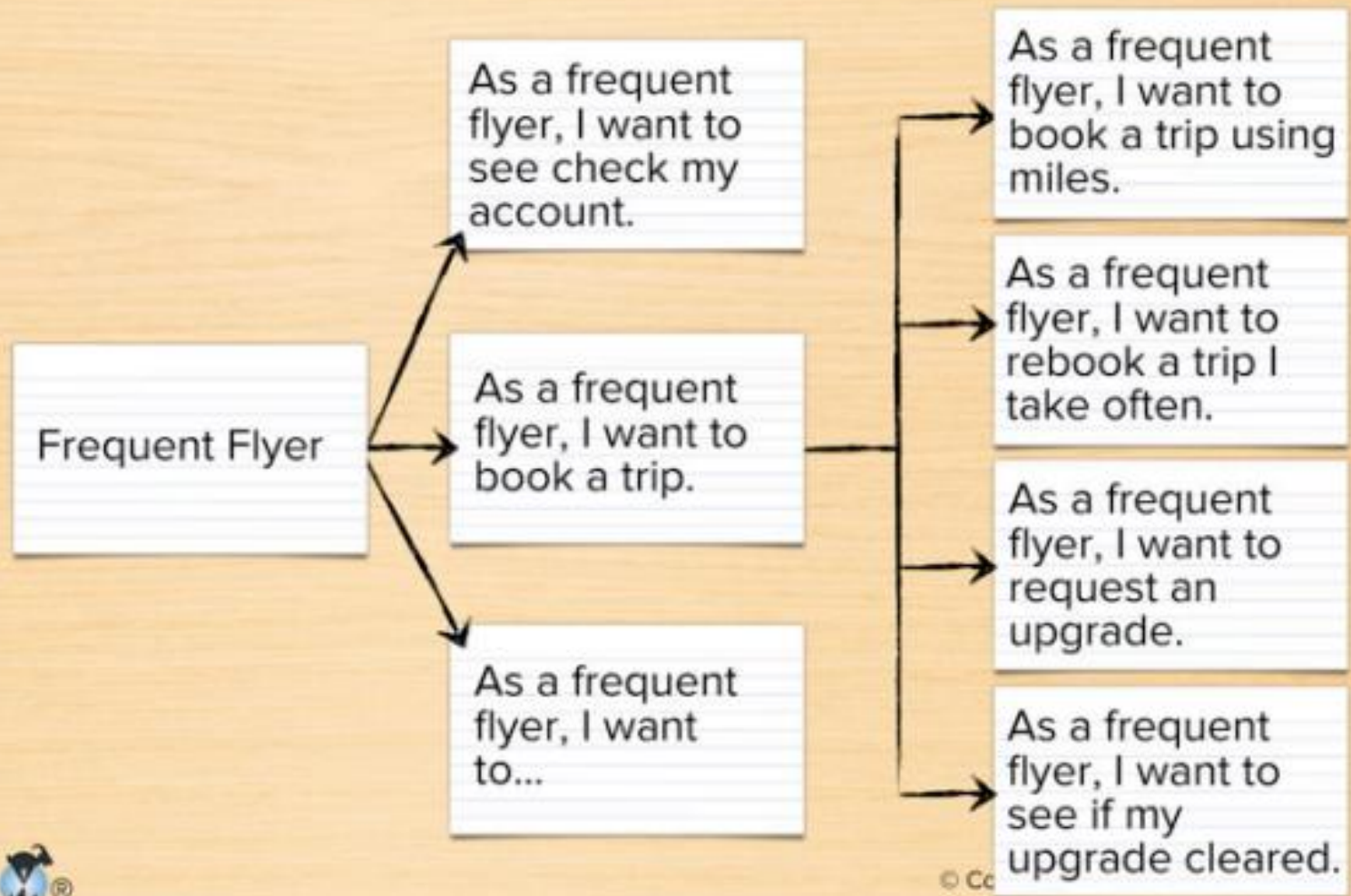
- ❑ Les US doivent être petites
- ❑ Fonctionnalités initiales: macroscopiques (*vision doc*)

➔ Effectuer un **Split**



(source : <http://succeeding-with-agile.blogspot.fr/>)

Start with epics and iterate



(source Mike Cohn: www.mountaingoatsoftware.com/)

User Stories ou Use-Cases ?

- Les UC sont utilisés en UP, les US en Agile

➔ Quel mode de représentation du besoin choisir ?

UC



■ avantages

- ✓ Représentation synthétique du besoin puissante
- ✓ Structure (packages, relations includes & extends,...)
- ✓ Formalisme et détails (UC/scénarios/steps, bordereau détaillé)

■ inconvénients

- ✓ Granularité > durée d'une itération → inutilisables tels quels en Agile

■ avantages

- ✓ Légers, faciles à lire
- ✓ Rapidement rédigés

➔ bien adaptées dans certains contextes (métier connu, client présent sur site, équipe réduite, ...)

■ inconvénients



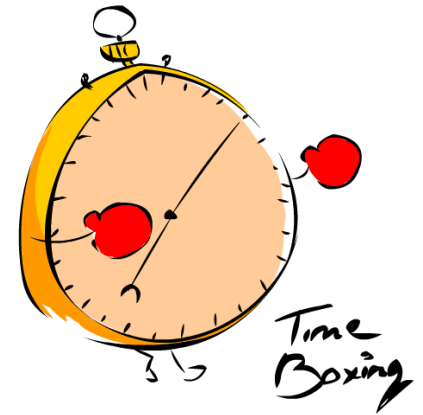
- ✓ Peu de formalisme, de **détails** et de traçabilité
- ✓ **Manque de contexte** : dans quelles circonstances les actions sont réalisées, quel est l'objectif plus large de l'utilisateur au-delà de l'US
- ✓ Vision éparpillée: des dizaines de post-it sur un mur...

- **Rendre les deux modes compatibles:**
 - ✓ Modélisation des besoins macroscopiques = UC Model
 - ✓ Splitter les UC en US: **Chaque scénario du UC est une US** (ou partie de scénario – Cf. techniques de Split)

Time Boxing

- ❑ Dans un projet agile tout doit « tenir » dans une durée fixée:
 - toutes les réunions, chaque point de l'odj
 - la réalisation de tous les travaux
 - les itérations, les releases, ... le projet.

- ❑ Schedule Wins (vs Scope)
 - Les dates de fin d'itération/release doivent toujours être tenues.
 - Le périmètre couvert s'adaptera.



(Angela Quero 2008)

“

Time Boxing

=

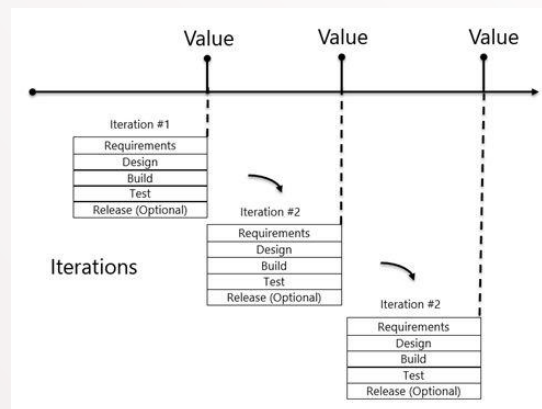
*auto-discipline qui
véhicule l'idée que le
temps est important*

❑ Le Time Boxing favorise:

- La régularité et l'atteinte d'objectifs de court terme
- La livraison rapide du produit
- L'élimination des activités improductives car dans un timing serré on n'a le temps de se consacrer qu'à l'essentiel

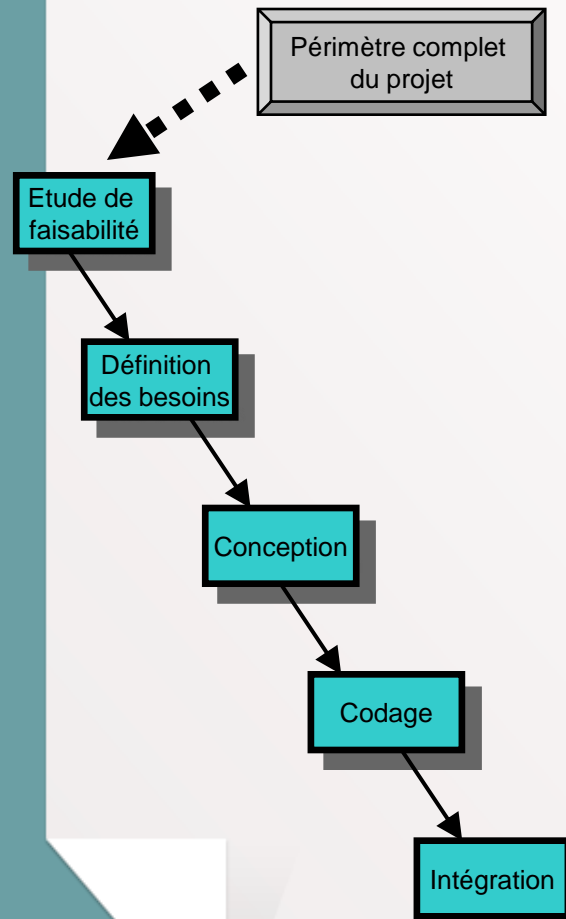


Parallélisme des activités & Emergence des besoins

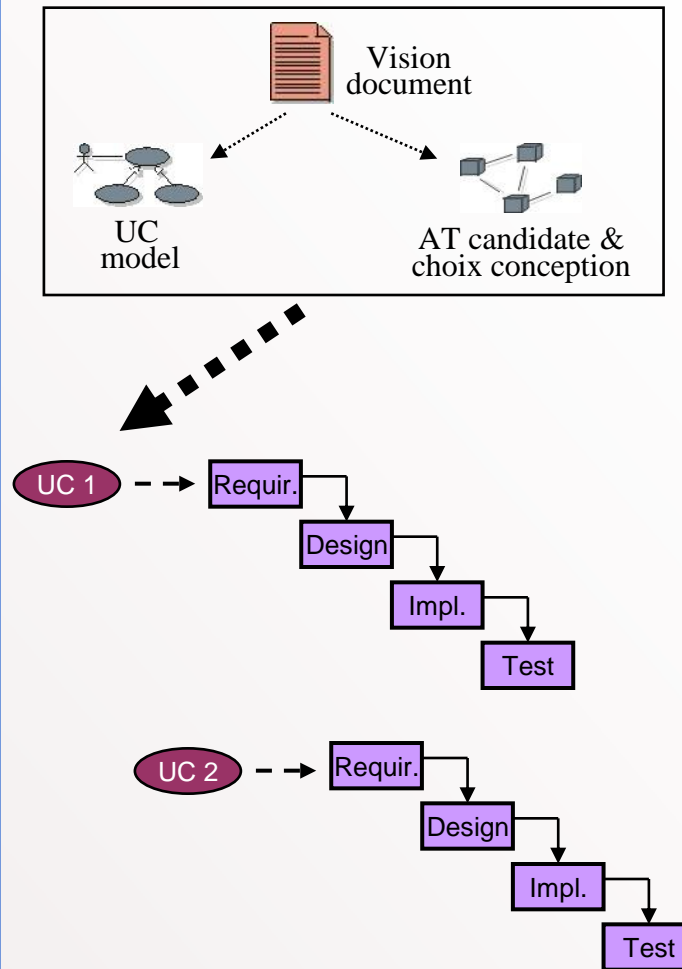




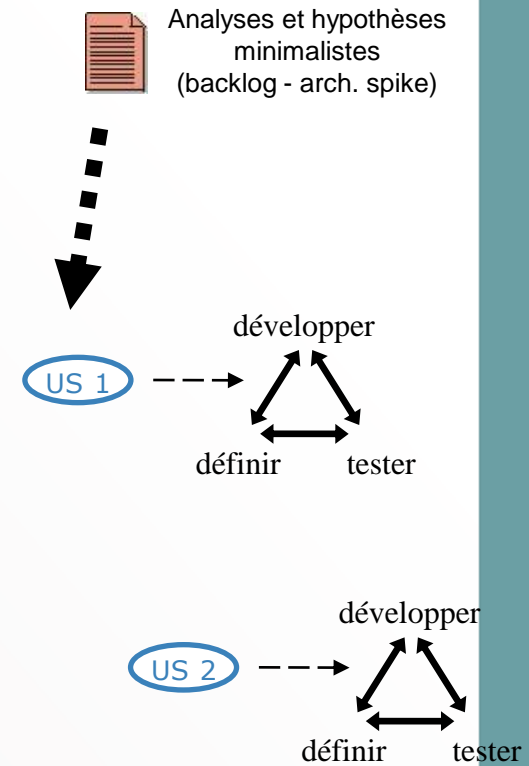
Waterfall



UP



Agile





Breakdown Structure

❑ Waterfall : **Work Breakdown Structure**

Le travail est décomposé par activités.

- séparées les unes des autres
- portant sur la totalité des fonctionnalités

❑ Agile : **Feature Breakdown Structure**

Le travail est décomposé par fonctionnalités (US).

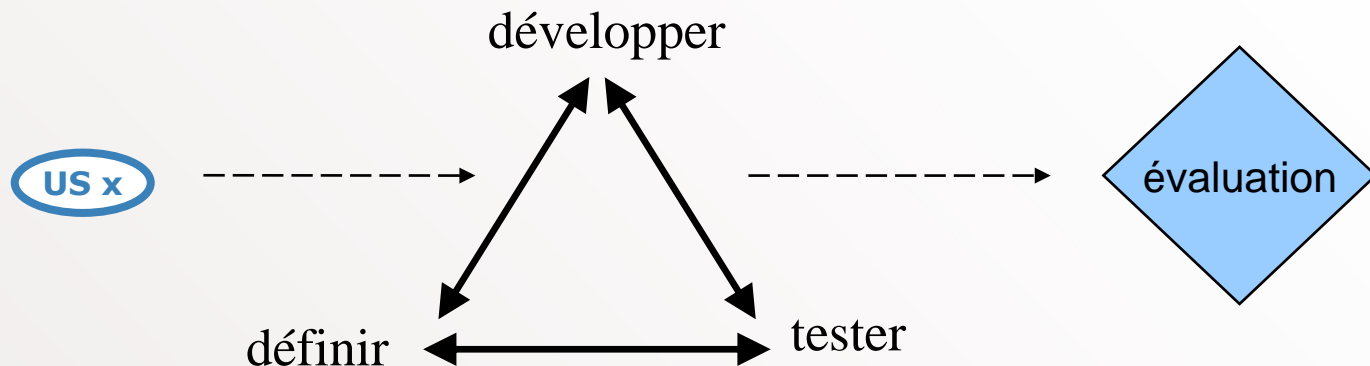
Les activités s'exécutent en parallèle ce qui rend leur distinction minimale.

❑ RUP : **Feature + Work**

Le travail est d'abord décomposé par fonctionnalités (UC) puis par activités clairement dissociées

Définir, Développer et Tester en parallèle

- ❑ Les activités requises pour livrer une US sont:
 - Classiques (décrire fonctionnellement, analyser, concevoir, implémenter, tester à tous les niveaux TU, TI, TA) mais...
 - **Peu dissociables** car elles s'influencent mutuellement, de manière itérative





❏ Exemple

- a) on définit la US de manière superficielle (orale + schéma au paperboard), on en tire un premier design qui remet en cause partiellement la définition.
- b) Les tests d'acceptance et les tests unitaires sont écrits sur cette base.
- c) Un premier développement est réalisé; les tests unitaires échouent. Le code est ensuite plusieurs fois refactoré (allers-retours modèle-code) jusqu'à passer les tests unitaires.
- d) les tests d'acceptance sont passés, ils échouent et conduisent à faire évoluer la définition de la US (évolution fonctionnelle) et/ou à modifier le code ou la conception (bugs).



“

En poussant l'agilité à l'extrême on réalisera toutes ces activités en parallèle.

Seule l'évaluation du PO est nécessairement réalisée après ces activités, une fois la US complétée



Evolution des spécifications et de l'architecture

❑ Waterfall : « Big Up Front Design »

On investit des mois de travail dans des spécifications exhaustives écrites tout en amont du projet.

Ces choix sont ensuite figés.



❑ UP : « Architecture and Specs Grow »

Des choix structurants pour l'ensemble du projet sont effectués initialement :

- Création d'un UC model organisant l'ensemble des besoins
- Identification d'une architecture candidate
- Choix de conception fondamentaux (langages, ...)

Ces choix sont ensuite affinés et enrichis tout au long du projet :

- Description détaillée des UC, enrichissement du UC model
- Validation de l'architecture à travers le prototype architectural
- Design progressif des UC s'inscrivant dans l'architecture définie



❏ Agile : « Architecture and Specs Emerge »

- L'approche est très empirique :
 - Spikes et hypothèses minimales tout en amont...
 - ... validés par une première version du système
 - améliorer le produit par adaptations successives, rapides et incrémentales
 - Chaque nouvelle fonctionnalité et sa solution sont ainsi détaillées au « **dernier moment responsable** »
- ➔ ne pas spécifier par avance des besoins qui ne verront jamais le jour.



“

Les besoins réels, les spécifications, les décisions architecturales, les choix de conception (et le produit) **émergent de façon continue**, tout au long du cycle de vie du projet.



Emergence et parallélisme : inconvénients

- ❑ Certaines situations de forte complexité rendent impossible l'émergence continue et/ou le parallélisme complet :
 - une problématique métier complexe
 - ✓ requiert une réflexion fonctionnelle importante en amont afin d'anticiper sur les difficultés, de planifier les différents cas.
 - l'absence de framework pré-établi ou
 - l'emploi de technologies non maîtrisées
 - ✓ requiert de prendre des décisions architecturales ou faire des choix de conception structurants pour l'ensemble du projet avant l'implémentation de toute US.

[illegible]



❑ Waterfall

Les spécifications sont ultra détaillées et décrites avec un formalisme élevé

❑ UP

Le formalisme est élevé (divers modèles UML). Le niveau de détail est modulable mais souvent très élevé

❑ Agile

Les spécifications sont détaillées au strict minimum nécessaire permettant l'implémentation. Elles sont souvent exclusivement orales ou écrites de manière informelle :

- US cards
- Schémas au tableau (photographiés si besoin)
- Commentaires du code

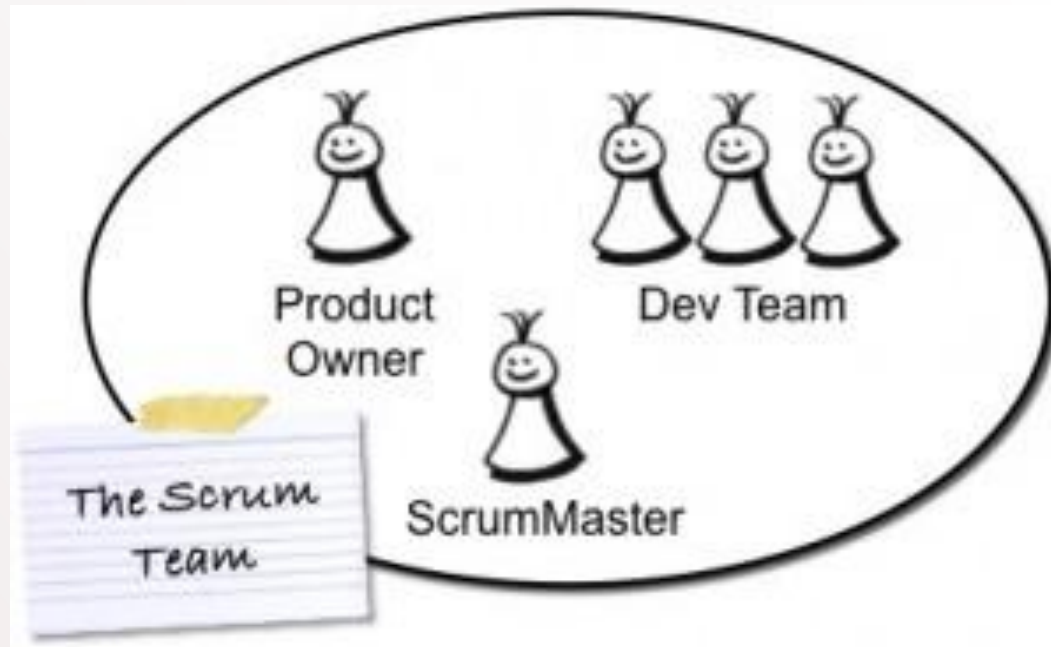


Faible formalisme : inconvénients

- ❑ Métier complexe → impossible de décrire les besoins à travers quelques phrases (US cards)
- ❑ US Cards + Code → pas de vue d'ensemble du besoin et de la solution
- ❑ Manque de pérennité
 - Pas de trace précise des réflexions complexes menées
 - Complexifie les évolutions futures du système
- ❑ Equipe non totalement colocalisée → nécessaire d'écrire des spécifications

Equipe colocalisée et polyvalente

- Rappel: Equipe type agile



(source : <http://www.romanpichler.com/blog>)

Colocalisation

- ❑ Mindset agile = Valider rapidement des US.
 - **Collaboration permanente:** réponse immédiate aux interrogations des coéquipiers.
 - Hyper **réactivité** au **changement**.
 - Interaction de toutes les compétences requises.
- ➔ **toutes les ressources** nécessaires doivent être présentes sur le même site, **y compris le PO.**

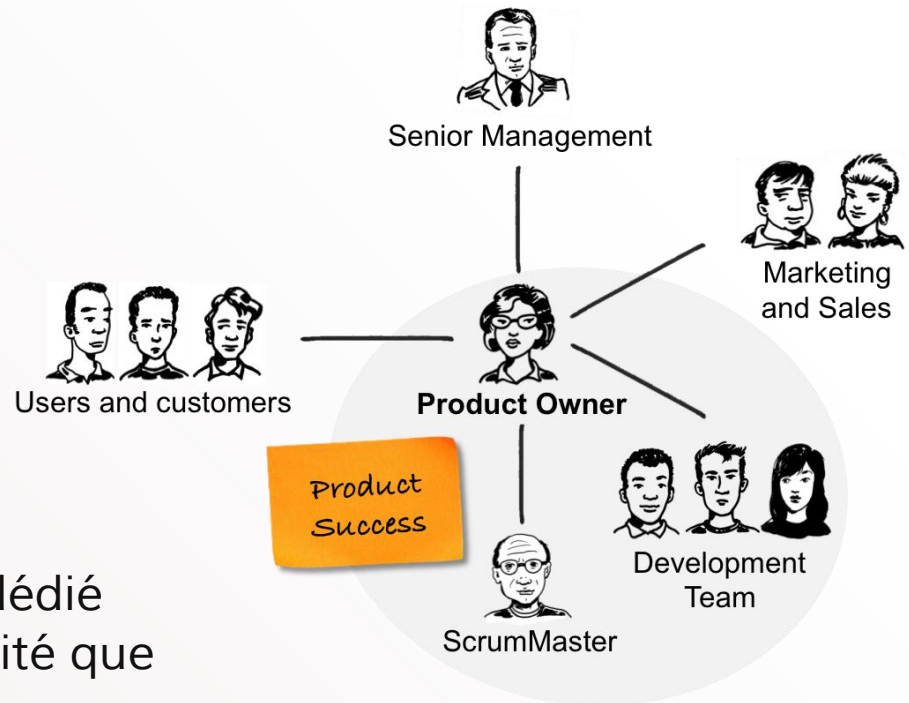
Rôles du PO sur le projet

❑ La présence du PO (ou du client) est nécessaire. Ses activités sont :

- Présentation « face à face » des US aux TM → diffuse la connaissance métier
- Participation aux spécifications et test-cases
- Réponse immédiate aux questions et suggestions de l'équipe
- Validation progressive des travaux (exécution des test-cases, acceptation des US)
- Planification des releases et itérations
- Démonstration du produit et validation des itérations

Le PO dans l'organisation

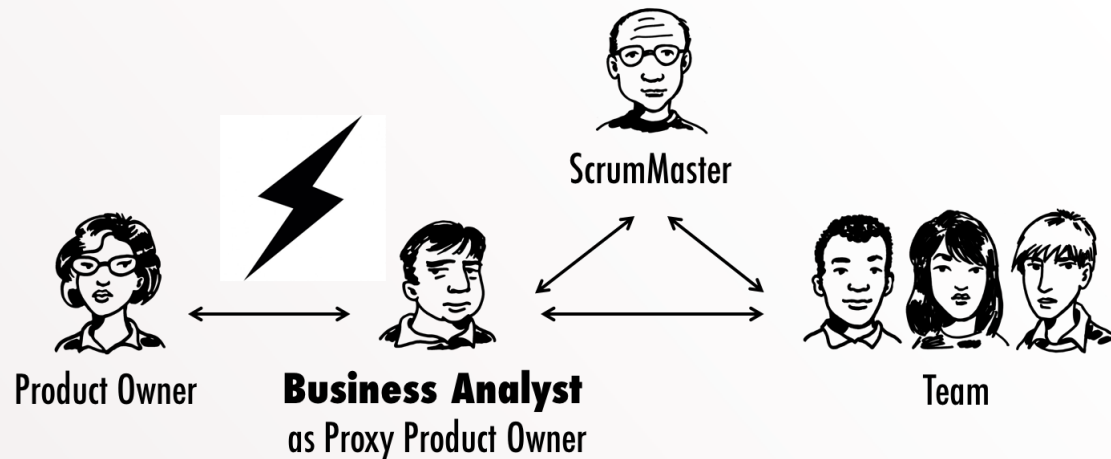
- ❑ Le PO travaille avec les clients et les décideurs pour définir, communiquer et prioriser les besoins.
- ❑ Il peut être :
 - un **chef de produit** marketing (produit destiné au public)
 - un **key user** (produit à usage interne)
 - un **représentant utilisateur** dédié (appartenant à la même entité que l'utilisateur)



(source : <http://www.romanpichler.com/blog>)



- Si aucun PO ne peut être présent sur site avec l'équipe projet, ou seulement à temps partiel, son rôle pourra être joué par un **Business Analyst**.



(source : <http://www.romanpichler.com/blog>)

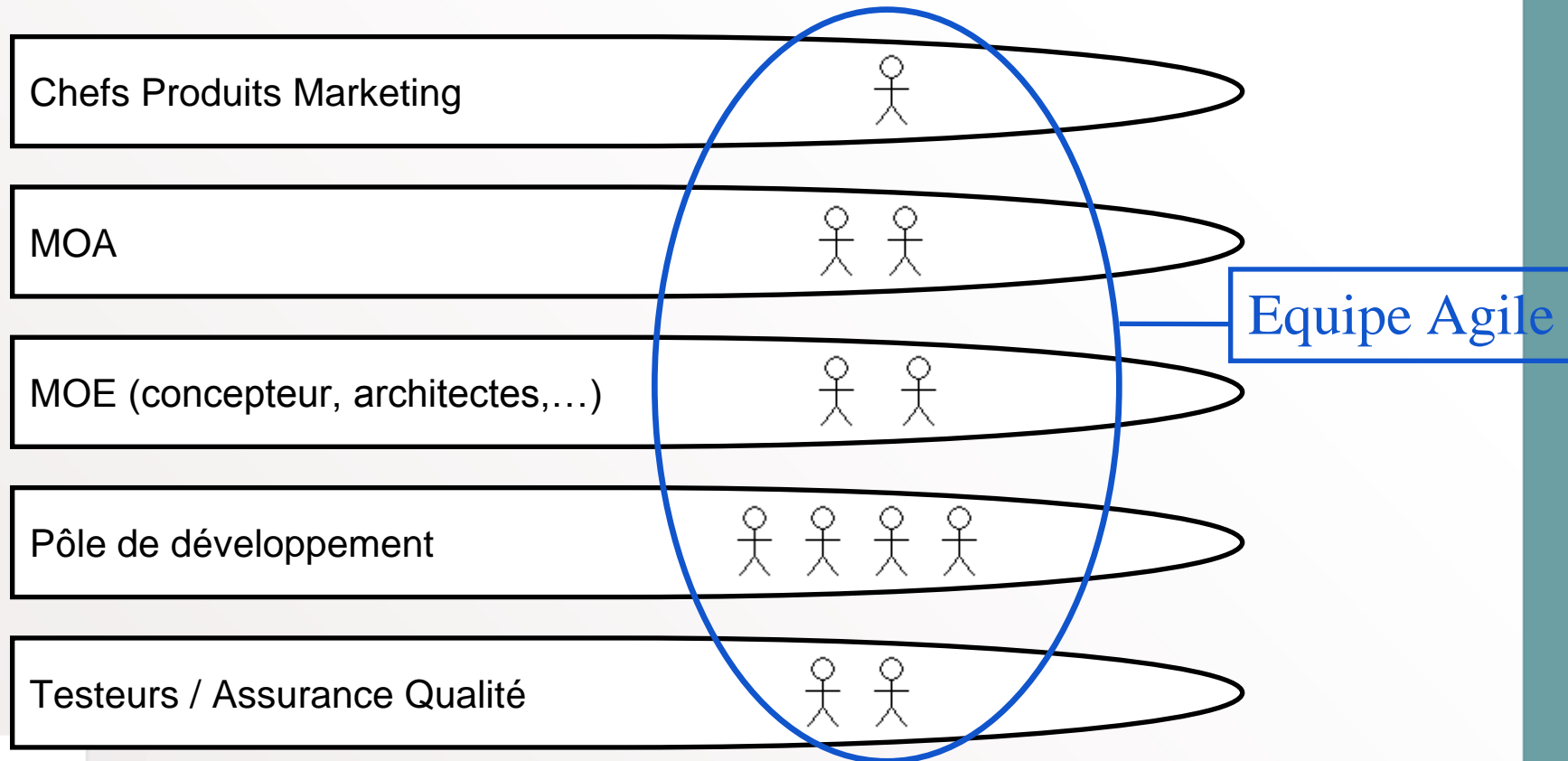
- Celui-ci devra être en contact très fréquent avec le PO. Les activités du PO pourront être réparties sur ces deux membres.

Polyvalence

- ❑ Distinguer des rôles précis parmi les TM est inefficace
 - Equipe de spécialistes d'une seule discipline (conception, développement, tests, business analysis, etc.)
➔ goulets d'étranglement.
 - Si chaque TM peut effectuer toutes ces disciplines et réaliser seul une US ➔ **souplesse et vitesse max.**
 - Au minimum: « **spécialistes généralisants** » = Expert d'une discipline et compétents dans au moins une autre.

Absence de cloisons fonctionnelles

- ❑ Organisation traditionnelle = TM cloisonnés. Chacun appartient à un silo fonctionnel et/ou organisationnel.



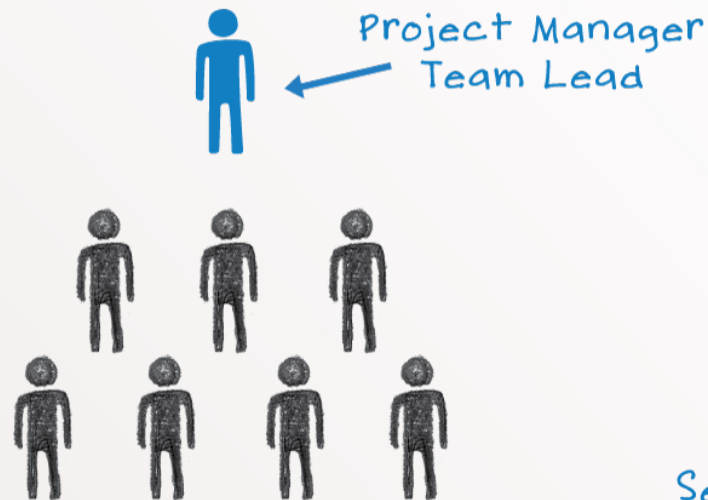
- Agile = Regrouper toutes les compétences, les distinctions de fonctions et d'organigramme sont secondaires.

Scrum Master

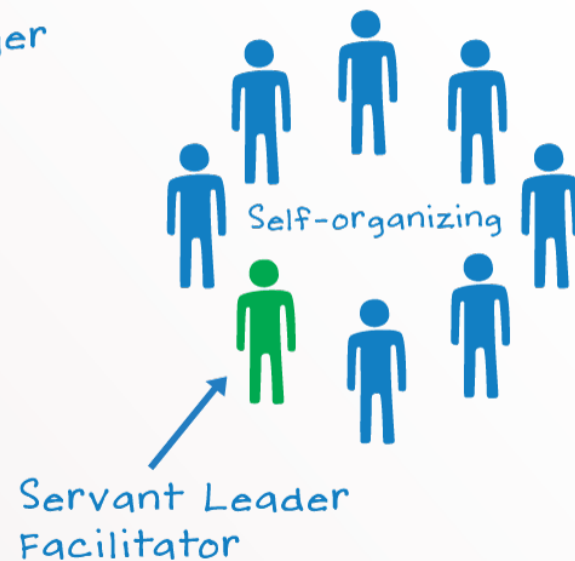
- Peu interventionniste, laisse l'équipe prendre les décisions. Son rôle consiste à:
 - ✓ Faciliter le travail de l'équipe, l'aider à accomplir sa tâche
 - ✓ Rappeler et **enseigner les règles et principes agiles**
 - ✓ **Eliminer les obstacles** qui bloquent le progrès de l'équipe, (relations avec le top management, obtenir ressources, liens avec les autres projets, etc...)

- Rôle très différent d'un Chef de projet: approche plus classique (type open UP), plus directif, prend les décisions importantes.

Traditional Teams



Agile Teams





Equipes auto-organisées

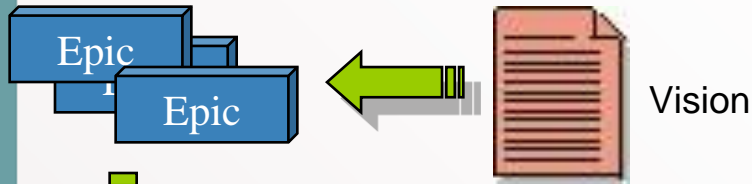
- ❑ La direction se contente de donner les directions et de lever les obstacles.
- ❑ En contrepartie l'équipe prend la pleine responsabilité du respect des dates et de la qualité du produit livré.
- ❑ Chaque membre de l'équipe participe activement à l'estimation des charges, à la planification, à la discussion sur le contenu des US.

Planification multi-niveaux

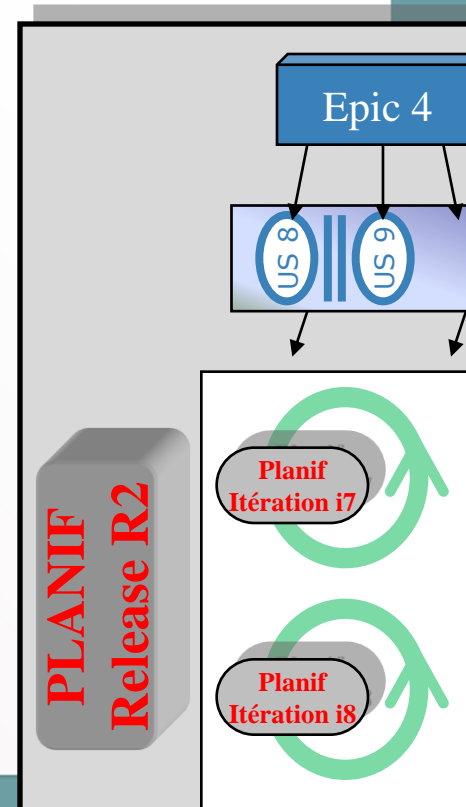
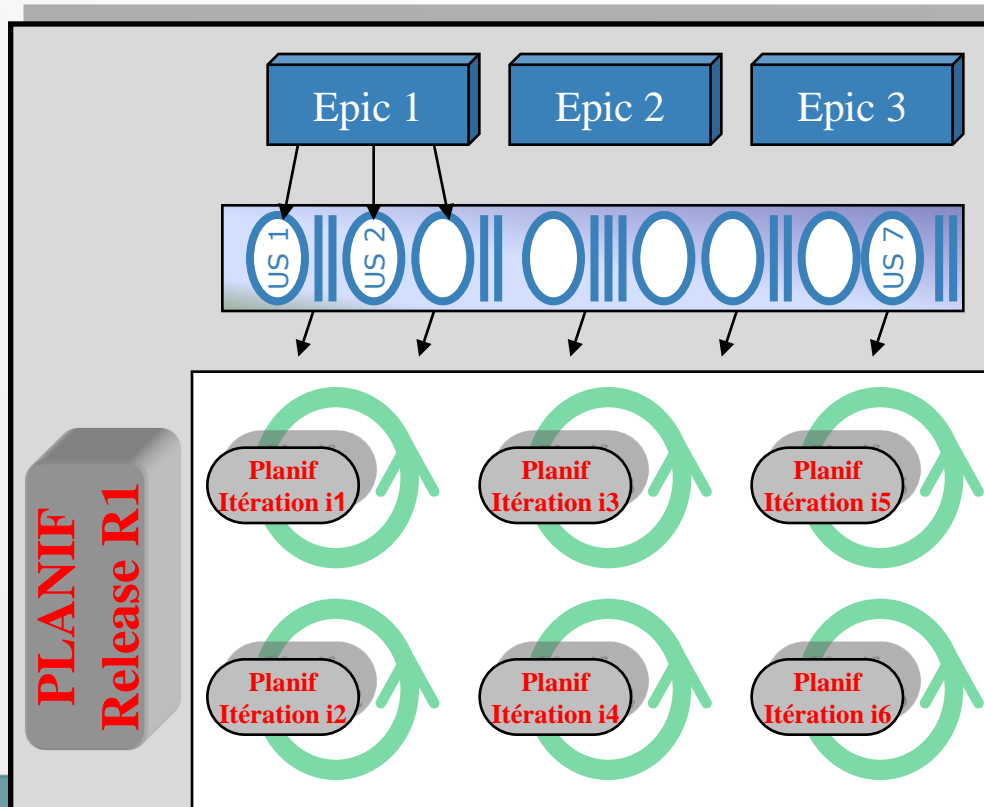
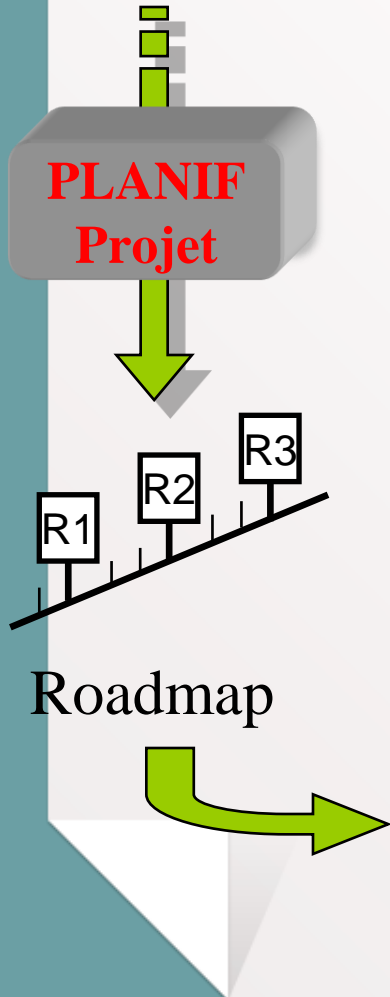


- ❑ La planification agile est revue fréquemment et à 3 niveaux différents:

Niveau de planification	Fréquence	Granularité	Précision
Projet (Product Roadmap)	1 à 2 fois par an	Objectif/Feature	très général
Release	1 fois par release	Epics→US	assez approximatif
Itération	1 fois par itération	US→tâches	précis



La planification agile s'effectue à 3 niveaux différents



❑ Précision croissante

- Cette planification est imprécise à LT (inévitablement) mais très précise à CT.
- Revue fréquemment ➔ sans cesse affinée.
- Schedule wins : les dates communiquées sont sûres, fournissent des objectifs et des repères.

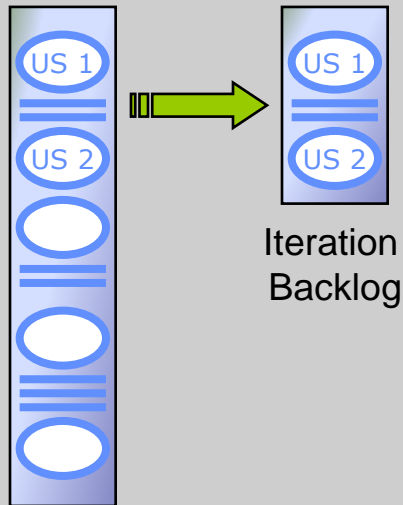
Voir chapitre III a – Planification et Estimation des charges

Cycle de vie d'une itération

½ journée

½ journée

PLANIF Itération

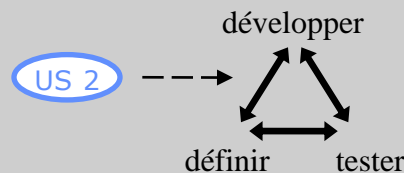
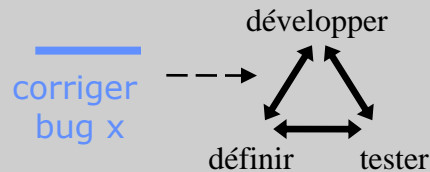
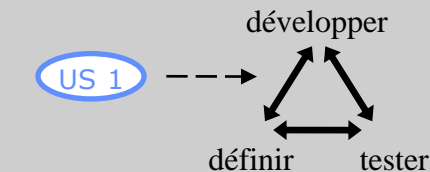


Iteration Backlog

Release Backlog

- discuter les US
- décomposer les US en tâches
- estimer charges
- prioriser
- s'engager sur un périmètre

EXECUTION Itération

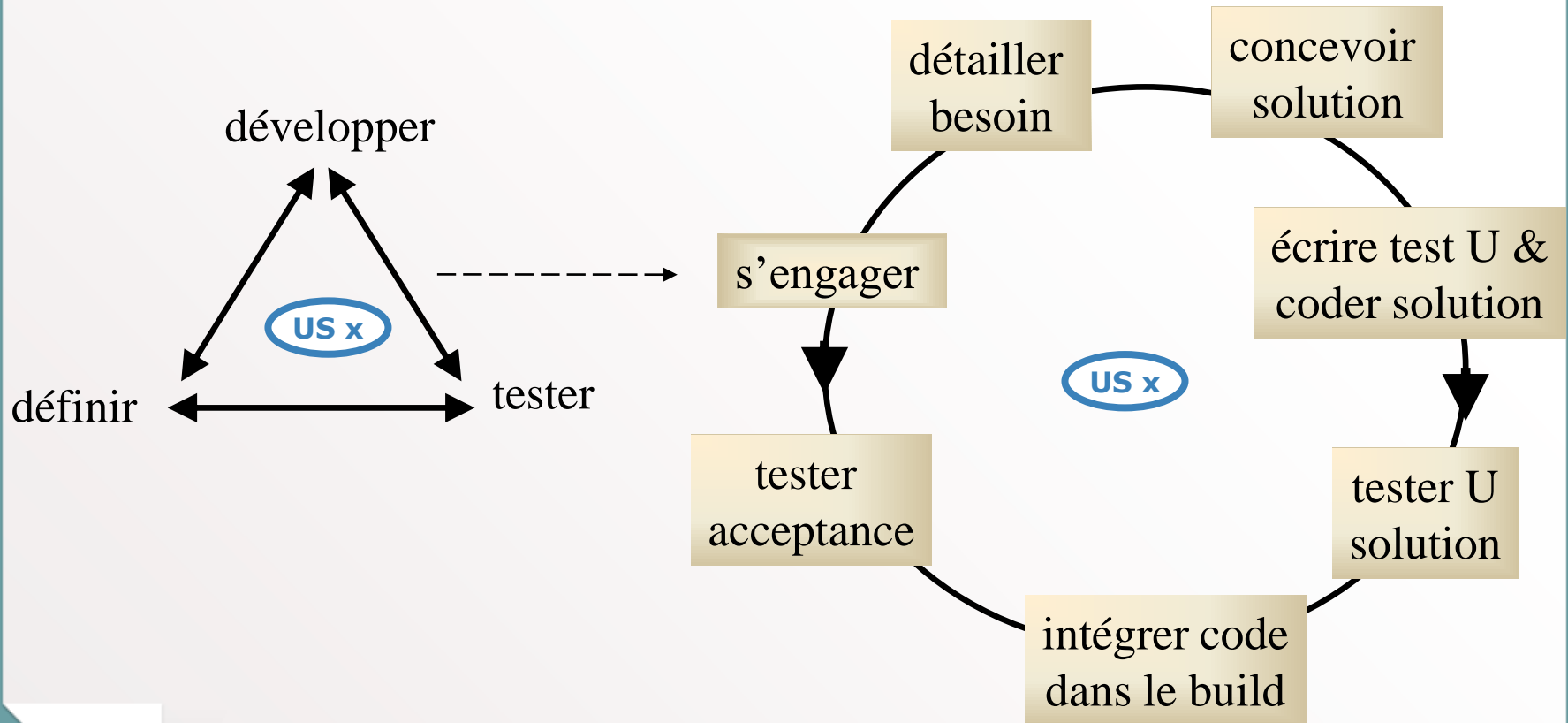


Mesurer le progrès

REVUE Itération

- Démonstration auprès de :
 - product owner
 - clients
 - décideurs
 - utilisateurs
- Feedback
- Valider l'itération
- auto-évaluation
- Màj du backlog (items non finis ou nouveaux)

Réaliser une US: détail des disciplines



Tests et Intégration continus

« Never break the build –
The system always runs »

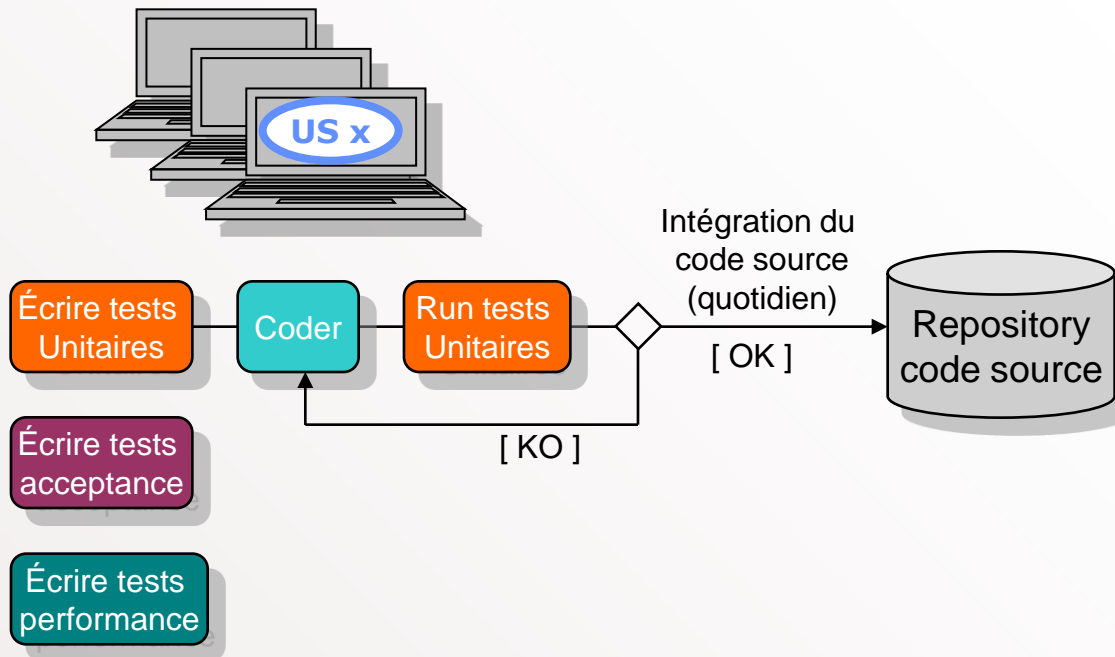


- ❑ But : Avoir en **permanence** une **version** du système disponible pour déploiement/livraison.
- ❑ Pouvoir y intégrer immédiatement toute nouvelle fonctionnalité terminée sans le dégrader.

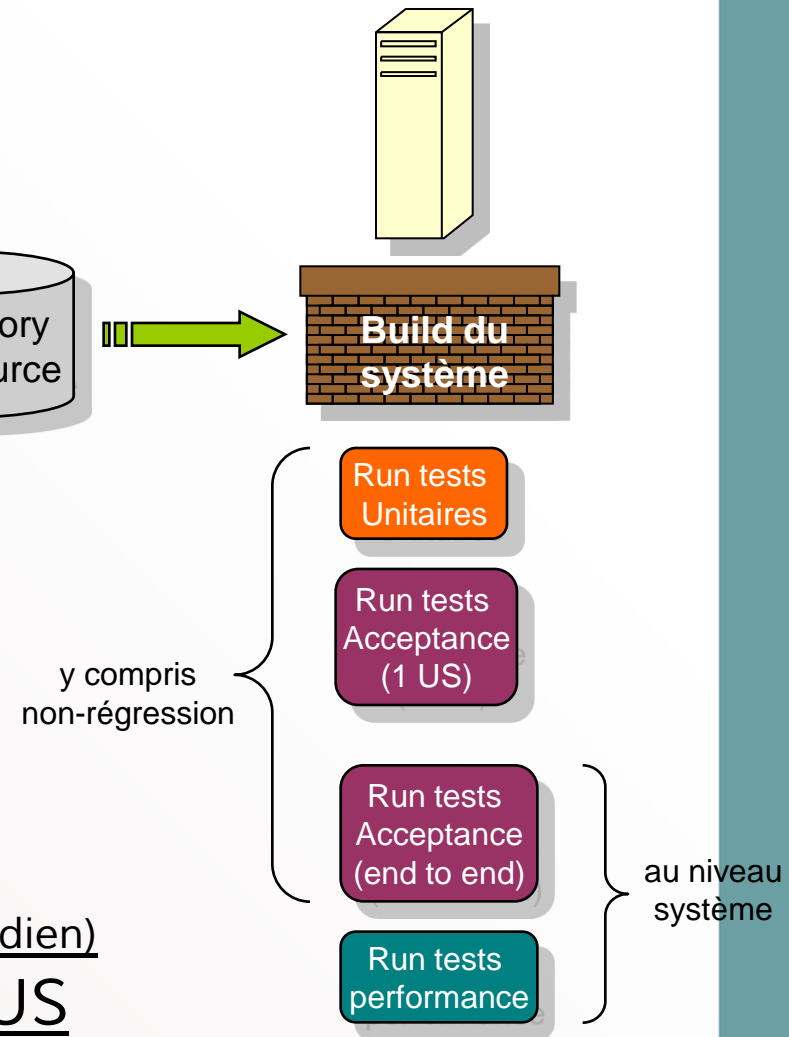
TDD

- ❑ Test Driven Development : Le test du code est écrit avant le code lui-même
 - contraint le développeur à comprendre le comportement du système et la façon de le valider avant d'écrire le code
 - garantit l'objectivité de la validation
 - garantit l'existence des TU

Environnement de Développement



Environnement d'Intégration



Processus continu (au moins quotidien)
de test et d'intégration des US

- ❑ Chaque US (ou correction de bug) traitée dans une itération subit l'ensemble des tests.
 - Tous tests OK ➔ US intégrée.
 - Moindre KO ➔ le build précédent est maintenu.
- ❑ Le seul type de **code valable** = **du code testé**, à tous les niveaux, qui fonctionne et intégré au build.
- ❑ Repository commun: l'intégralité du code source est toujours accessible, collectif, sauvegardé.
- ❑ Le build automatique génère un report informant l'équipe du succès/échec et des tests échoués.

“

Intégration Continue

Le système est incrémenté
quotidiennement et est
**toujours potentiellement
livrable.**

Automatisation

❑ Les tests sont automatisés au maximum



- Vitesse et facilité d'exécution des tests
- Les TNR (tests de non-régression) peuvent être systématiques
 - ✓ les fonctionnalités intégrées sont forcément testées à plusieurs reprises
 - ✓ La qualité du logiciel est accrue



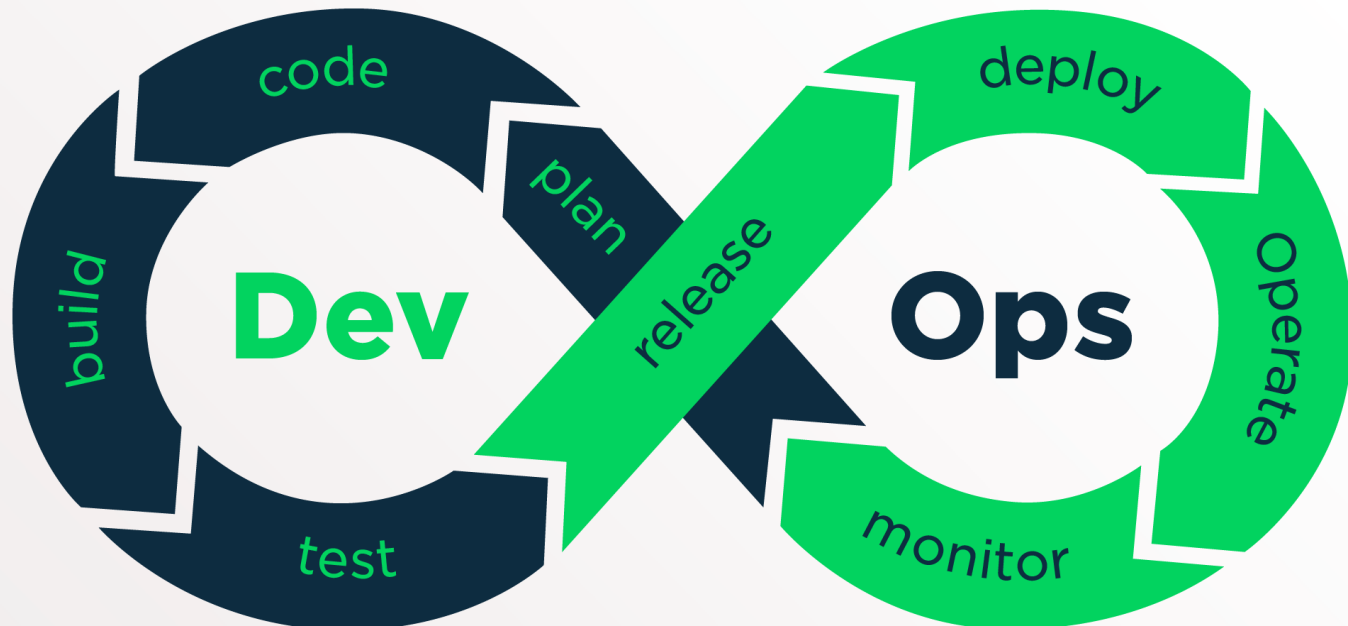
❑ L'automatisation totale coûte cher ⚠



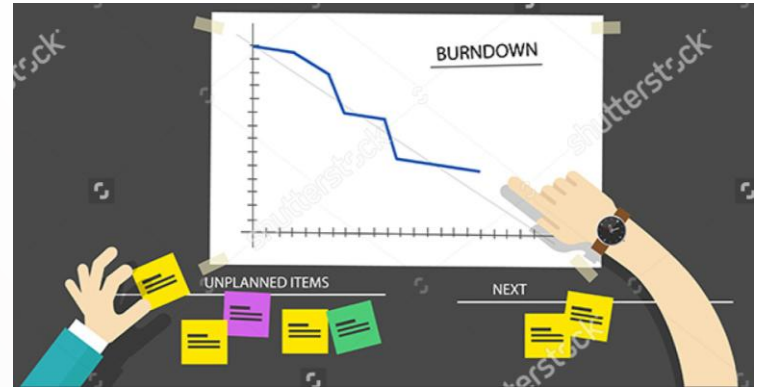
- souvent les TI et TA restent manuels ➔
- tous les tests ne peuvent être réalisés au cours de l'itération prévue et glissent vers la fin de la release
- Prévoir en fin de release une itération « harden » consacrée à la préparation de la release, incluant:
 - ✓ les tests non finalisés
 - ✓ des tests de performance en conditions de production
 - ✓ l'élimination des derniers bugs (mineurs)
 - ✓ quelques refactorings de code
 - ✓ réalisation de livrables (manuel utilisateur, doc d'install, packaging, ...)

DevOps

- ❑ Rapprochement des équipes de Développement (Dev) et d'exploitation (Ops – opérationnels)
- ❑ Appliquer les principes de l'agilité à la production
 - Le « Continuous Integration » peut être étendu au « Continuous Delivery » (déploiement continu du S.u.d.)



Mesure de l'avancement



- ❑ Avancement traditionnel =
 - Atteinte des jalons conformément au planning établi
 - Livraison de documents

- ❑ Avancement Agile =
 - Capacité à s'adapter au changement
 - Valeur réellement apportée au client par le Produit

“

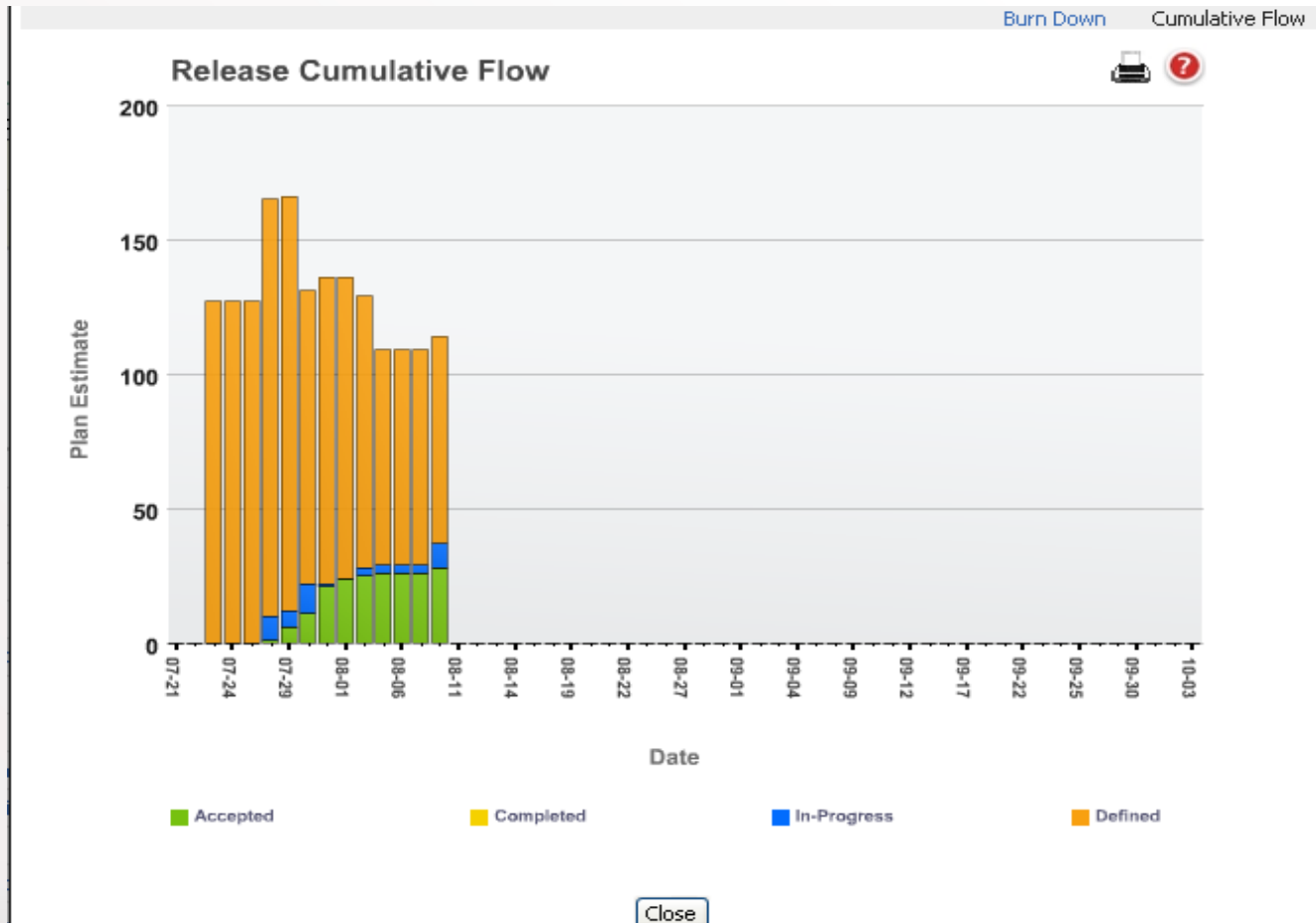
Avancement Agile

Work Items du backlog
totalement **finis** et
acceptés par le client

(WI finalisé à 99% = aucune
valeur client).

Représentation de l'avancement

□ Burndown ou burnup chart



Voir chapitre III b – Mesure de l'avancement



Auto-évaluation

- ❑ Les équipes agiles prennent la responsabilité de la réussite ou de l'échec du projet.
 - Elles sont responsables du choix et de la bonne application de la méthode

- ➔ Elles évaluent fréquemment leur performance au cours des rétrospectives d'itération et de release.

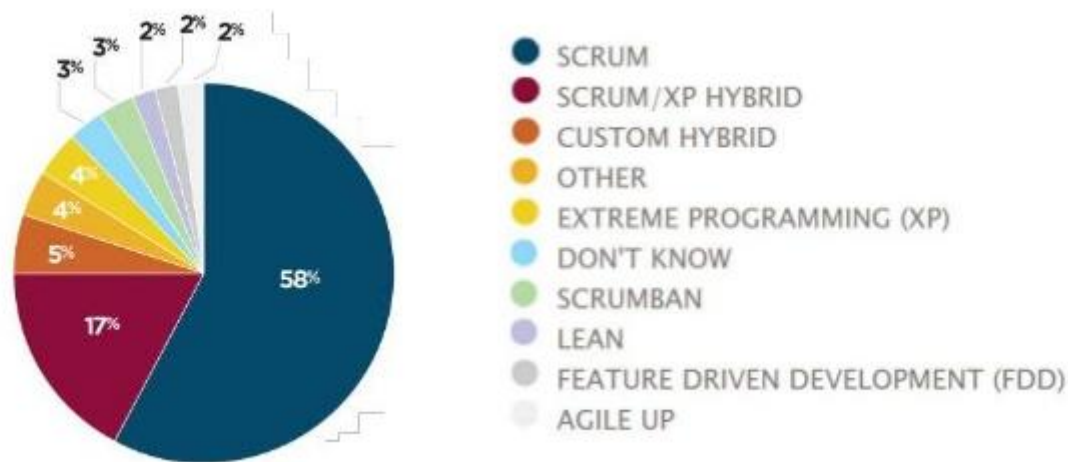


- ❏ Quantitativement (indicateurs)
 - Pourcentage de US acceptées
 - Nombre de bugs
 - Nombre de builds sans échec
 - Pourcentage de scénarios avec test automatique
 - ...

- ❏ Qualitativement
 - S'interroger objectivement sur ce qui a marché ou non au cours de l'itération/release,
 - Notamment la maîtrise de la méthode
- ➔ En tirer des enseignements pour les itérations à venir



4 – Adoption des Méthodes Agiles



Limites



Limites et faiblesses des méthodes agiles

❑ Taille de l'équipe

- les méthodes agiles sont parfaitement adaptées à des équipes de petite taille (< 10 personnes).
- Les utiliser à plus grande échelle requiert une adaptation.

❑ Présence du client

- Très efficace avec un PO présent à 100%. Mais sinon ?
- Produit destiné au public OU dizaines d'utilisateurs internes: Existe-t-il un product owner représentatif de tous?
- Globalement: existe-t-il un PO disponible, capable de prendre des décisions et d'exprimer correctement le besoin ?



- ❑ Equipe colocalisée
 - est-il possible de rassembler sur un même site des ressources appartenant à des entités voire des entreprises ou des pays différents ?
- ❑ Granularité des User Stories (cf [section correspondante](#))
- ❑ Emergence continue et parallélisme (cf [section correspondante](#))
- ❑ Faible formalisme (cf [section correspondante](#))
- ❑ Coût de l'automatisation des tests (cf [section correspondante](#))



Obstacles propres à l'entreprise

- ❑ Résistance du département « méthodes »
 - L'entreprise s'accroche à ses anciennes méthodes connues et rejette le changement

- ❑ Culture de l'entreprise
 - Le top management rejette l'absence de planning détaillé à LT, la présence de story cards et de burndown charts affichés au mur, des ressources passant beaucoup de temps à parler, etc...

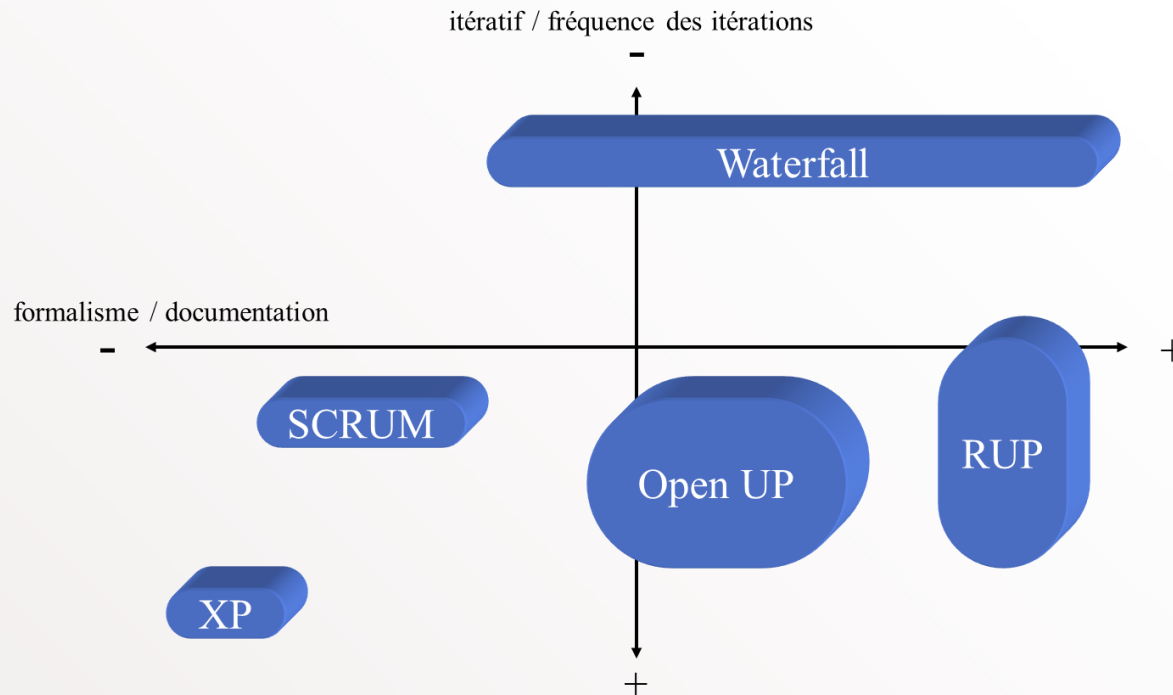


- ❑ Existence de procédures formalisées
 - L'entreprise institutionnalise le respect strict de chaque étape (revue et signature des specs), du plan qualité, etc...

- ❑ Contrats fixes
 - Le projet peut être soumis à un contrat (notamment forfait avec un fournisseur) qui implique la livraison de fonctionnalités fixes à date fixe

- ❑ Equipes distribuées
 - l'entreprise a pu délocaliser le pôle de développement

5 - Les Méthodes



Conclusion

UP / Agile: Méthodes flexibles

- ❑ Simples cadres à l'intérieur duquel définir son propre processus.
- ❑ Faire son choix parmi les pratiques proposées, selon les besoins, le contexte (taille, complexité, métier, technos, disponibilité du client, expérience de l'équipe, ...).
- ❑ Le choix de la méthode peut être un mix (eg XP + Scrum).



Toujours adapter la méthode au contexte