

Approche Agile

Les méthodes agiles

- 
- Cycle de développement **itératif**, **incrémental** et **adaptatif**.
 - En 2001, est né le **manifeste Agile - adaptatif**
<http://www.agilemanifesto.org/>.

Définition Agilité

« L'agilité est la capacité à favoriser le changement et à y répondre en vue de de s'adapter au mieux à un environnement turbulent. »

Jim Highsmith, un des signataires du Manifeste Agile, cité dans Aubry, C. (2014). Scrum, Le guide pratique de la méthode agile la plus populaire. Dunod, p3.

Manifeste pour les développements Agile

Nous
par la pratique et en aidant les autres à le faire,
reconnaissons la valeur des seconds éléments,
mais privilégions les premiers.

Les individus et leurs interactions plutôt que les processus et les outils

Des logiciels opérationnels plutôt qu'une documentation exhaustive

La collaboration avec les clients plutôt que la négociation contractuelle

L'adaptation au changement plutôt que le suivi d'un plan.

Equipe

Les individus et leurs interactions plutôt que les processus et les outils

- Equipe de développeurs soudée et qui communique.
- Equipe de développeurs de différents niveaux qui communiquent bien entre eux plutôt qu'une équipe composée d'experts fonctionnant chacun de manière isolée.

→ Communication

Application fonctionnelle

Des logiciels opérationnels plutôt qu'une documentation exhaustive.

- D'abord un logiciel fonctionnel.
- Importance du logiciel documenté (code) mais pas les documents produits relatifs au projet :
 - **Logiciel testé ET documenté pour pouvoir le maintenir.**
 - A minimiser : documents projet : rapport d'avancement, notes explicatives, planning revu et maintenu à jour en permanence, flux pour signature...

Collaboration

La collaboration avec les clients plutôt que la négociation contractuelle.

- Le client doit être impliqué dans le développement.
- On ne peut négliger les demandes du client.
- Le client doit collaborer avec l'équipe et fournir un feedback continu sur le logiciel.

→ logiciel adapté aux attentes du client.

Acceptation du changement

L'adaptation au changement plutôt que le suivi d'un plan.

- Réagir aux demandes de changement.
- Planification flexible.
 - Plan de développement au début du projet.
 - Revu et remanié à chaque nouvelle itération :
 - Permettre l'évolution de la demande du client tout au long du projet.
 - Prendre en compte les demandes d'évolution provoquées par les premières *releases* du logiciel.

Valeurs en résumé

- L'équipe collabore et accepte le changement pour une application qui fonctionne et répond aux besoins des utilisateurs.

Principes Agile (1)

- **Satisfaire le client** : plus haute priorité
 - Livrer **rapidement** des fonctionnalités à **grande valeur ajoutée**.
- **Accueillir les changements** de besoins, même tard dans le projet pour donner un avantage compétitif au client.
 - Conception orientée évolution.
- **Livrer fréquemment** un logiciel opérationnel avec des cycles de quelques semaines à quelques mois.
 - **Mise en production rapide d'une version minimale** du logiciel & ensuite **nouvelles livraisons incrémentales**.

Principes Agile (2)

- **Collaborer quotidiennement** entre utilisateurs (ou de leurs représentants) et développeurs.
- Réaliser les projets avec des personnes motivées, en fournissant l'environnement et le soutien dont elles ont besoin et en leur faisant confiance.
- Communiquer par des **conversations en face à face**.
- **Mesurer l'avancement par un logiciel opérationnel.**
- Garder un **rythme de développement soutenable** : maintenir indéfiniment un rythme constant.



Principes Agile (3)

- Rechercher **l'excellence technique et une bonne conception.**
- Rechercher la **simplicité** - càd minimiser la quantité de travail inutile.
- Laisser l'équipe **s'auto-organiser**, ne pas imposer de processus.
- Laisser l'équipe réfléchir aux moyens de devenir plus efficace & adapter son comportement.

Agile : adaptatif

Planification (1)

- **Planification itérative.**
- Exemple : le développement d'un projet est prévu sur 3 mois, itération de 2 semaines. (15 US, 4 T)

Planning Projet												
Spécifications techniques et fonctionnelles	US1, US7, US8, T1											
			US5, US6, US4, T2									
					T3, T4, US2							
							US9, US11, US12, US3					
								US10, US13				
										US14, US15		
Semaines	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
	Itération 1		Itération 2		Itération 3		Itération 4		Itération 5		Itération 6	

Planning initial : pas détaillé, on vérifie rapidement que l'on peut tout développer

Logiciel prêt à être livré

Planification (2)

- 
- T2

Planning mis à jour pour une seule itération (& avec client)

Agile : adaptatif

- **Fonctionnellement**, par adaptation systématique du produit aux changements de besoin.
 - La demande de changement est bienvenue mais
 - Changement n'est pas permanent.
 - Ne peut y avoir d'interruption intempestive du développeur qui travaille.
 - Engagement pour l'itération en cours n'est pas modifié.
 - On passe par une gestion des priorités et la demande de changement peut être différée.

Agile : adaptatif

- **Techniquement**, par remaniement régulier du code : **refactoring**.
 - Refondre le code source pour en améliorer la qualité.
 - Changer le design sans changer les fonctionnalités.
 - Fort encouragé dans les méthodes Agiles.
 - Outil de changement.

Deux méthodes agiles

eXtreme Programming

Scrum

2 méthodes

- eXtreme Programming
 - Premier projet en 1996.
 - En perte de vitesse.
 - A permis de développer des concepts et des pratiques qui sont utilisés dans les méthodes Agile.
- Scrum
 - S'est imposé.

<http://www.extremeprogramming.org/>
<http://xprogramming.com/index.php>
<http://extremeprogramming.free.fr/>

Quelques concepts et pratiques XP utilisés par d'autres

eXtreme Programming

Spike

- Une **spike solution** est une recherche de solution à un problème technique délicat ou à un problème de conception.
- C'est un programme simple qui permet d'explorer différentes solutions et qui ne concerne qu'un seul problème à la fois.
- C'est une façon de **réduire le risque** induit par ce problème.

User stories

- Une **user story** s'exprime sous la forme :
En tant que <utilisateur>, je veux/dois <fonctionnalité> afin de < objectif >
- Cette structure favorise la sélection de fonctionnalités pertinentes. Cela permet de clarifier à qui cette fonctionnalité pourra bénéficier, mais aussi dans quel but elle doit être développée.
- Une user story est utilisée en tant que spécifications mais également pour l'estimation du temps de développement et la planification.

User stories

- Format associé aux méthodes agiles.
- Description brève d'une fonctionnalité telle que vue par l'utilisateur.
- Format écrit **court**, laissant de la place à la discussion orale.
- Emergence rapide dans des ateliers collaboratifs.
- Grande **simplicité** et donc grande lisibilité.
- Histoire **implémentée en une seule itération** ! (découpe si besoin).

User stories

Exemple

ID

102

Nom

S'enregistrer

Histoire

- **En tant que** membre
Je veux m'enregistrer dans le système
Afin de pouvoir prendre un rendez-vous.

Validation

- Un membre a un nom, un prénom, une adresse complète (rue, no, boîte, code postal, ville) et un email.
- Toutes les données sont obligatoires, excepté la boîte.
- Un membre peut être un administrateur.

Poids

- (Effort estimé) - au moment de la création de la US, parfois estimation ou laissé vide et sera apprécié au moment du planning de l'itération

User stories

Exemple

ID

103

Nom

Choisir un médecin traitant

Histoire

- **En tant que** membre
 - Je veux** choisir un médecin traitant
 - Afin de** pouvoir suivre mon dossier médical.

Validation

- Le médecin traitant est un médecin parmi ceux qui ont manifesté leur intérêt à avoir une activité régulière.
- Un médecin traitant a indiqué des dates de disponibilité.

Poids

- (Effort estimé) 3

Questions ?

Vélocité

- La vélocité est un indicateur de pilotage. C'est la **capacité de l'équipe de développement à livrer les fonctionnalités prévues.**

Début itération

- Chaque US a un poids
- On planifie un nombre d'US équivalent à notre vélocité.

Fin d'itération

- Somme des poids des US **effectivement réalisées** = vélocité de l'itération.



Lire aussi

<https://www.aubryconseil.com/post/2007/la-velocite-nest-pas-une-mesure-de-productivite/>

Vélocité : exemple

Début itération

- User Story A : 2 points
- US B : 3 points
- US C : 3 points

Fin d'itération

- US A : terminée
- US B : non terminée
- US C : terminée

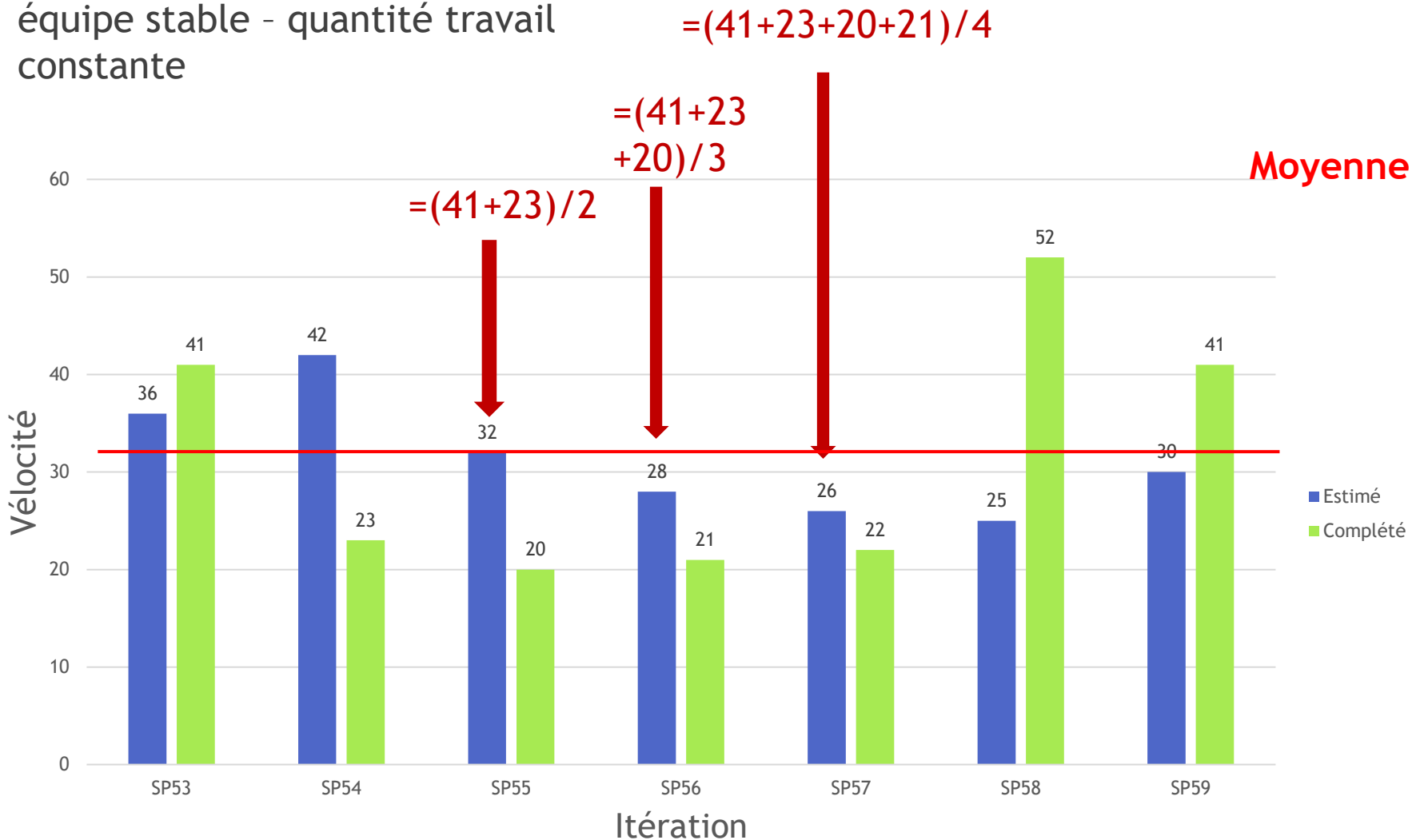
➔ **Vélocité = 5 points.**

Prévoir
itération
suivante :
5 points



La vélocité est une **constatation à posteriori**.

Calcul vitesse (pas de contexte) -
équipe stable - quantité travail
constante



Questions ?

<http://www.scrumalliance.org>
<http://www.scrum.org/>

Scrum

An Agile framework

Scrum

Méthode Agile

- Apporter plus de valeur aux clients et aux utilisateurs.
 - Maximiser la valeur ajoutée
 - En réalisant d'abord les fonctions à plus haute valeur ajoutée.
 - En changeant les priorités et même les fonctions.
 - ➔ Voir définition du sprint backlog.
- Apporter une plus grande satisfaction dans le travail.
 - Equipe auto-organisée.

Scrum

Scrum fournit un **cadre** pour le développement d'un produit complexe.

Scrum est adaptatif :

- Idée : il est impossible de définir tout dès le début : les spécifications peuvent changer, des outils ou technologies inconnus entreront en jeu, etc..
- Pour s'adapter aux changements, les travaux à faire sont ajustés à la fin de chaque itération.

Cadre

Cadre très léger :

- 3 rôles.
- Itérations.
- Réunions au début et à la fin de chaque itération.
- Mêlée quotidienne.
- Backlog de produit.
- Peu de production documentaire.

Rôles

1. Product owner

- **Product Owner : directeur de produit**
 - Il représente le client et les utilisateurs dans l'équipe.
 - Il a la vision du produit et l'autorité pour donner les priorités aux requirements du client.
 - Il est responsable du **Product Backlog**.
 - Il est responsable du résultat auprès des autres parties.
 - C'est le canal de communication avec les parties prenantes.

Rôles

2. Scrum master

- **Scrum Master : gestionnaire**
 - Il **facilite** l'application de Scrum dans l'équipe.
 - Il s'assure que la méthodologie soit respectée et provoque les changements organisationnels nécessaires à cela.
 - Il supprime ce qui pourrait interrompre les équipes et les protège des interférences extérieures.
 - Il s'assure de l'amélioration des pratiques et de l'organisation du travail.
 - C'est le **coach** de l'équipe.

Rôles

3. l'équipe de développement

- **Equipe**

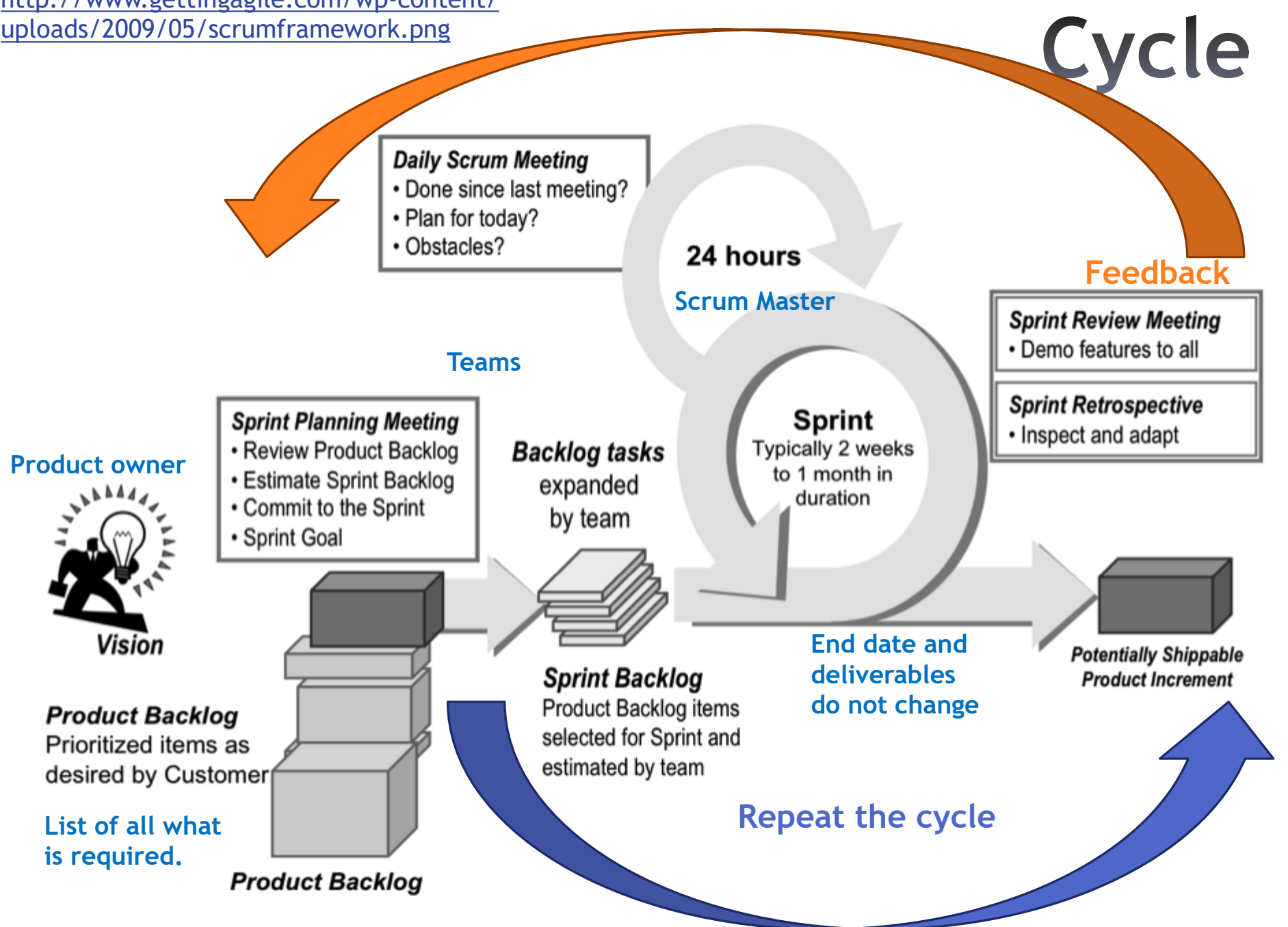
- Elle est composée de **l'ensemble des individus** participant aux activités de développement,
- Ces individus ont des compétences transversales,
- C'est une équipe performante et **autoorganisée**,
- Elle est centrée sur les livraisons.

Sprint

- **Sprint : unité de temps** qui permet de rythmer les développements, correspondant à une **itération** :
 - Sprint de durée courte (maximum un mois).
 - Pas de chevauchement.
 - Rythme régulier : tous les sprints de même durée.
 - Pas de changement de date de fin si le développement n'est pas fini.
 - Fixation de la quantité de travail à produire en fonction de la durée et de la taille de l'équipe.
 - ➔ budget fixe (si la taille de l'équipe est stable mais ceci est recommandé).

Backlog & version

- **Product Backlog** : liste, ordonnée par priorité, des fonctionnalités pour le produit, définie par le directeur de produit.
 - Les nouvelles demandes ou corrections de bugs sont ajoutées dans le product backlog.
- **Sprint Backlog** : liste des fonctionnalités qui sont développées dans le sprint.
- **Version** : plusieurs sprints peuvent être nécessaires pour développer une version du produit.



Réunion de début de sprint

(Sprint planning)

Planification du Sprint :

- Direction : ScrumMaster.
- Définition de l'objectif du sprint (Sprint Goal).
- Analyse du haut de la liste du « ProductBacklog ».
- Définition du « SprintBacklog » en fonction de la capacité de l'équipe et de la priorité des tâches.
 - Pour cela, il faut avoir estimé le poids des user stories :
 - Complexité, longueur, risques de complication.
 - Division éventuelle.
 - Consensus sur le poids.
 - Comparaison avec vélocité (moyenne de la vélocité des 3 derniers sprints, par exemple).

Mêlée journalière

Scrum Meeting :

- **Objectif** : savoir ce que chacun fait, se synchroniser, mesurer l'avancement, s'entre-aider.
- **Rapide tour d'avancement, équipe debout.**
- **Mise en place** :
 - Réunion une fois par jour à heure fixe.
 - Mise en commun des apports de chacun.
 - Partage des difficultés rencontrées.

Mêlée journalière

Scrum Meeting (suite):

- **Réponse aux 3 questions :**
 - Qu'as-tu **terminé** depuis la précédente réunion?
 - Que penses-tu pouvoir **terminer** d'ici la prochaine réunion?
 - Quels **obstacles** rencontres-tu en ce moment? (risque de ne pas respecter le sprint backlog)
- **Ajustements possibles**
 - Scrum Master détermine l'avancement par rapport aux engagements. Si nécessaire, ajustements.

Réunion de revue

(sprint review)

Réunion de revue :

- Démonstration des nouvelles fonctionnalités (en interne avec l'équipe).
- **Feedback** du directeur de produit.
- Redéfinition des priorités du ProductBacklog.
- Identification des user stories à traiter lors du prochain Sprint.

Rétrospective

Rétrospective :

- Prise de recul sur l'itération qui s'est terminée.
 - Identification des améliorations potentielles (processus, méthode de travail).
 - Détermination du « comment » réaliser ces améliorations :
 - Ce que l'on aimerait mettre en place.
 - Ce que l'on souhaite arrêter.
 - Ce que l'on souhaite continuer.
- ➔ Cette rétrospective permet donc d'améliorer le fonctionnement de l'équipe et d'augmenter la satisfaction de ses membres.

Cycle & Sprint

A l'intérieur d'un Sprint

- Développement d'un produit partiel :
Spécifier, analyser, concevoir, développer, tester, documenter et intégrer pour chaque US

Pendant le cycle de sprints

- Spécifications continues : le client précisera ses demandes au moment où elles sont développées.
- Conception continue : l'architecture va évoluer pendant la vie du projet.
- Tests dès le premier sprint.

Quand est-ce fini?

- Les Sprint se répètent jusqu'au moment où :
 - Il y a eu assez de fonctionnalités développées dans le ProductBacklog ou
 - Le budget est épuisé ou
 - La date limite du projet est atteinte.
- Les fonctionnalités à plus haute valeur ajoutée ont été développées !

Périodes

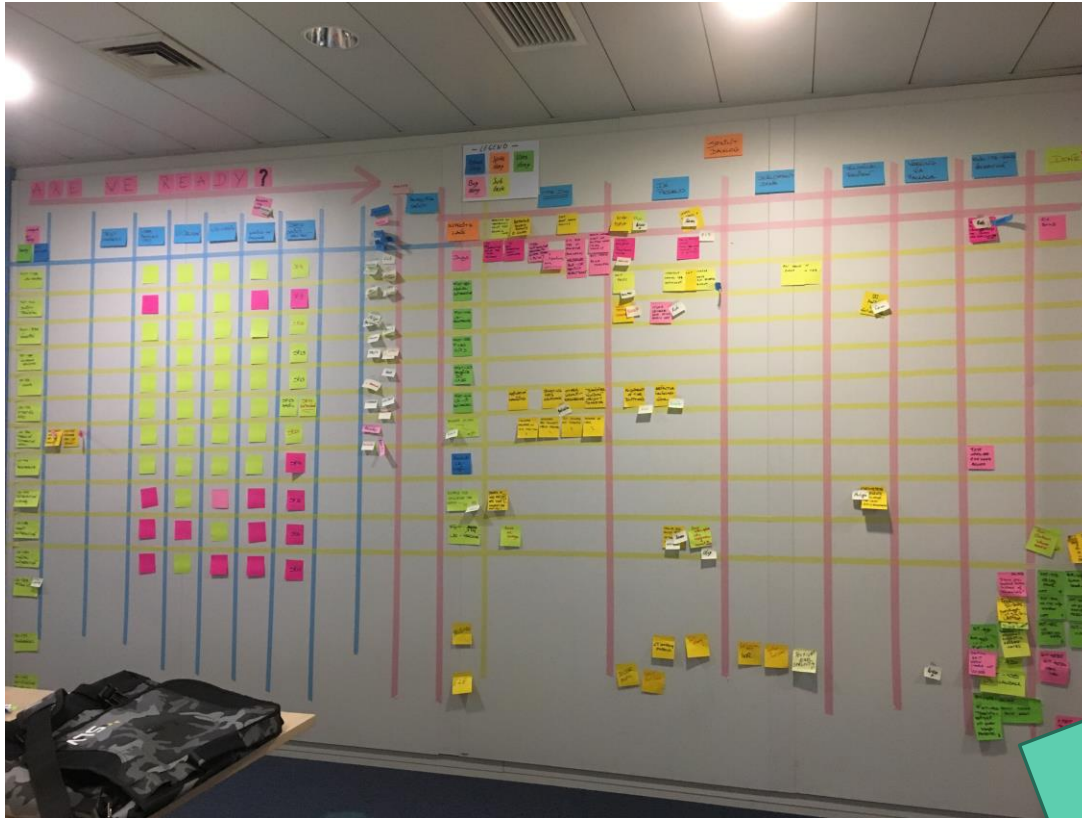
Il y a quand même 3 périodes en Scrum !

- Avant le premier sprint, après discussions avec le client et remise de prix :
 - Définition backlog initial.
 - Définition de la vision.
 - Constitution de l'équipe, mise en place de l'environnement.
 - Maîtrise de l'architecture.
 - Première planification générale.
- **Les sprints : période centrale .**
- Une période de clôture: tests d'acceptation sur tout le produit/projet/release; déployer l'environnement de production, écrire les manuels et donner la formation.

Exemples

Au quotidien

Scrum board : répartition des tâches



ARHS
Developments
Belgium – client
Proximus

Stage observation
Ronsmans Thomas
2016-2017

Couleurs ont une
signification : par
exemple, les jaunes
représentent les
tâches du front-end

Stage d'observation de
Frédéric Hubert
2017

Scrum





Une illustration

Description de l'organisation du travail

Stand Up Meeting

Stand up meeting

- Tous les matins à 10h
- Chaque développeur présente un résumé des tâches qu'il a accomplies le jour précédent et formule les remarques qu'il a à faire sur celles-ci
- Le responsable répartit les nouvelles tâches
- Un dashboard est mis-à-jour .

	To Do	On Going	Done	Test	Prod
Legacy					
User story 1					
User story 2					
User story 3					
User story 3					
...					

Parking


Description de l'organisation du travail



Organisation en Sprints de 2 semaines

Sprint
2 semaines

Mercredi	Sélection des scénarii utilisateurs Développement
Jeudi	Développement
Vendredi	Tests
Lundi	Correction des bugs
Mardi	Développement
Mercredi	Développement
Jeudi	Développement
Vendredi	Tests
Lundi	Correction des bugs
Mardi	Mise en production

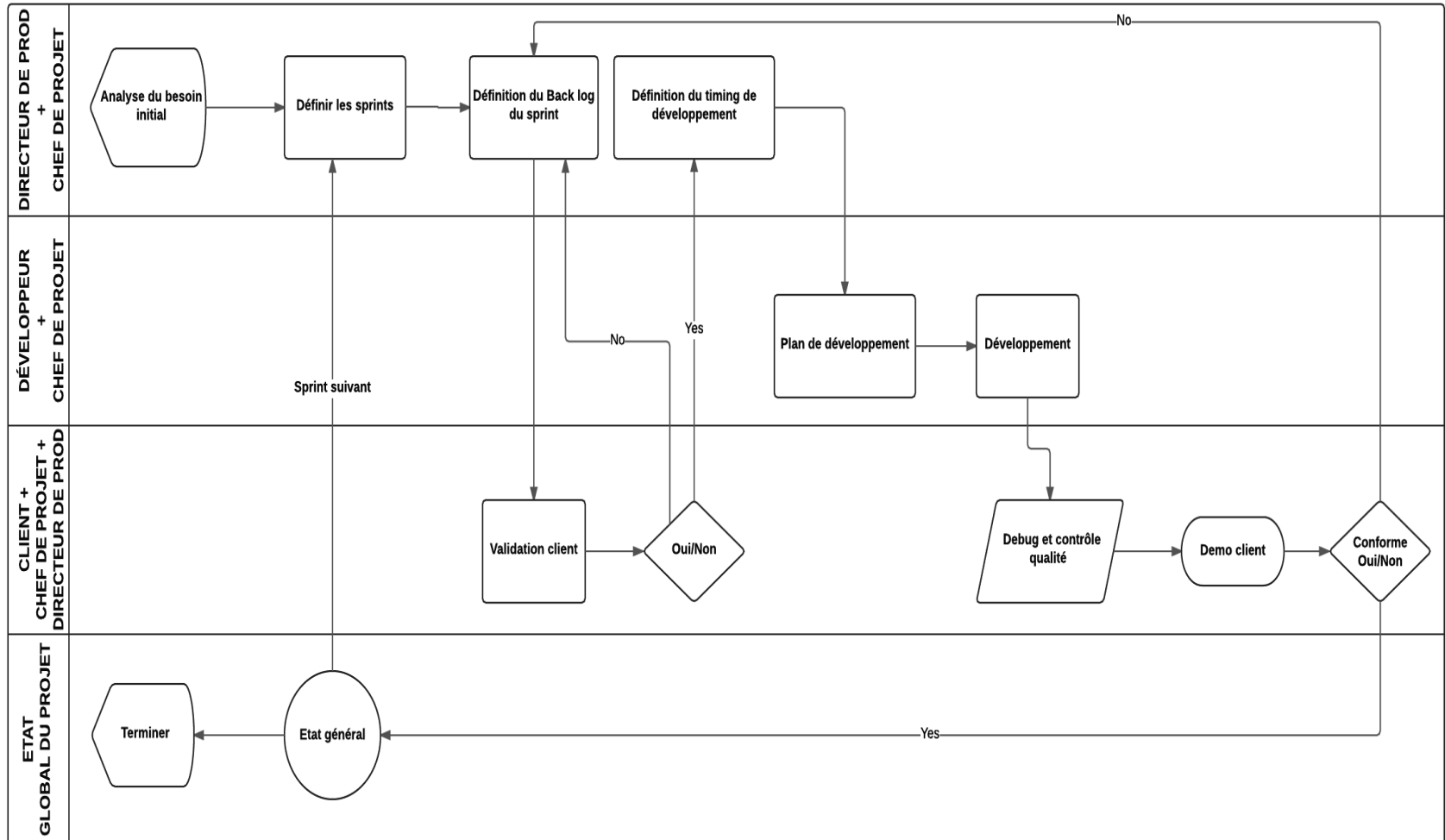
Stage d'observation de
Vanmoortel Nolan
2017

Scrum

Une illustration

PROCESSUS DE DÉVELOPPEMENT

Vanmoortel Nolan | ITDM sprl



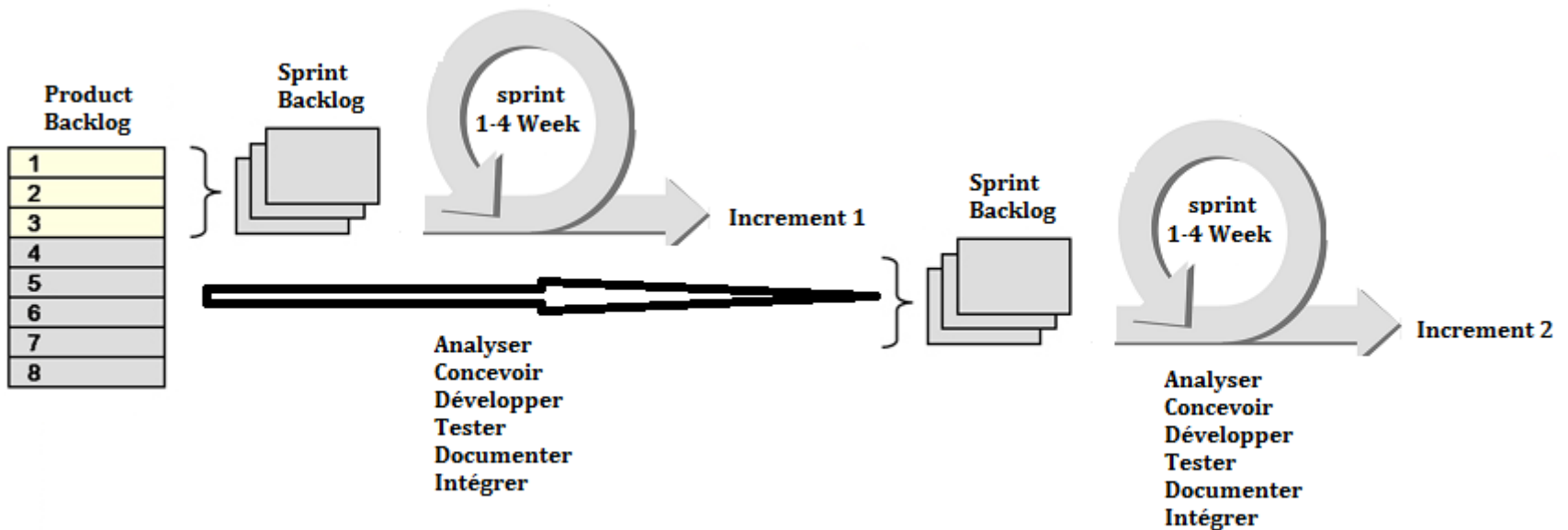
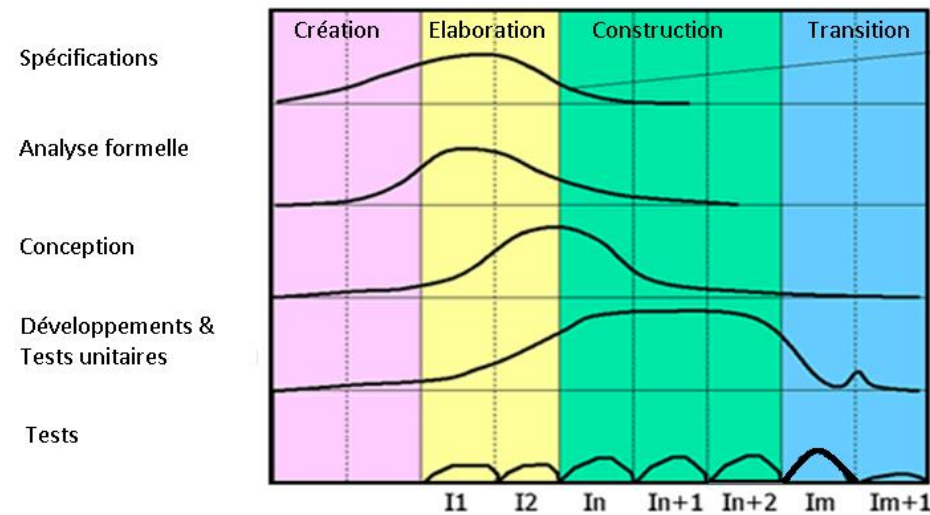
Livrable	Quand
1. Rapport d'analyse initiale	S3
2. Implémentation architecture : revue du code en séance	S5
3. Revue du code en séance	S8
4. Démo d'avancement en séance	S8
5. Logiciel en v1.0	S10
6. Demande de changement + complément d'analyse + Rapport + Démo	S12
7. Evaluation individuelle architecture	S13

Projet AE - PU ou Scrum ?

Comparaison PU et Scrum

PU

versus Scrum



PU

versus Scrum

- Définition complète des objectifs du projet.
- 4 grandes phases.
- Planning : date de fin du projet définie.
- Outputs très bien définis.

- Backlog.
- Chaque itération couvre le cycle complet pour les US.
- Product owner détermine quand le projet est fini.
- Output : code fonctionnel et documenté, backlog à jour.

Les étudiants font cela seuls ; de nombreuses informations reprises ci-dessous ont déjà été présentées pendant l'exposé des méthodes.

L'ingénierie logicielle : choix d'une méthode

Cascade, V ou Y :

- Une approche très contrôlée du développement
 - Planification précise.
 - Méthodes d'analyse et de conception.
 - Documentation.
 - Simple et facile à comprendre.
 - Attente entre 2 étapes.
 - Manque d'adaptabilité.
 - Problèmes vus à la validation (client).
- Beaucoup de temps passés sur la façon de développer un système avant le développement lui même.

Itératif : Pros

- Première version du système fournie rapidement.
- **Risques d'échec diminués**
 - Découverte des problèmes assez tôt.
 - Parties importantes développées en premier.
 - Limitation des coûts, en termes de risques, à une itération.
 - Gestion de la complexité et rythme de développement soutenu grâce à des objectifs clairs et à CT.
 - Avancées évaluées au fur et à mesure de l'implémentation.
- **Progrès visibles rapidement.**
- **Tests et intégration** se font de manière « continue »

Itératif : Cons

- **Définition de l'itération** : délicat, demande du temps, risqué.
- **Lourd** à mettre en œuvre.

Inadéquat pour les petits projets.

Agile

- Avantages :
 - Méthodes focalisées sur le développement.
 - Méthodes basées sur une approche itérative.
 - Livraison rapide & feed-back des clients.
 - Développement d'applications dont les exigences changent.
- Inconvénients :
 - (Souvent) Agile poussé à l'extrême & aucune documentation.
 - Difficulté à mettre en œuvre dans :
 - Affectation des priorités.
 - Simplicité dans les changements additionnels.
 - Implication intense des développeurs.

Agile (2)

- **Frontière ténue** entre l'application de la méthode et le « n'importe quoi » qui peut advenir quand on travaille sans méthode.
- Comme l'Agilité peut accueillir et parfois même favoriser le changement, certaines équipes pensent qu'ils peuvent changer tout tout le temps.
- Certains managers pensent qu'il sera plus facile de faire travailler les équipes en urgence par du travail supplémentaire non prévu.

Scrum contesté

- Application « sauvage » de Scrum :
 - Membres d'une équipe sont dans différents lieux.
 - (Outsourcing) membres de l'équipe viennent d'organisations différentes.
 - Membres de l'équipe n'ont pas le pouvoir de prendre les décisions.
 - Les décisions relatives au projet sont prises sans consulter l'équipe et elle dispose d'un minimum de liberté pour déplacer les « user stories » dans un sprint.

➔ Application doit être conforme à la méthode.

Quel modèle choisir ?

- Il n'existe pas un modèle idéal, un seul modèle à appliquer en toute situation.
- Un compromis en fonction du contexte semble idéal.
- Adopter un cycle de vie est déjà une preuve de maturité pour l'entreprise.

Comparatif

	Discipline	Agile
Taille (de l'équipe)	<ul style="list-style-type: none"> Nécessaire si grosses équipes Difficile à adapter aux petits projets 	<ul style="list-style-type: none"> Petites équipes Forte dépendance au mode de connaissance tacite qui limite la taille DANGER
Criticité (pertes en cas de défaut)	<ul style="list-style-type: none"> Idéal pour gérer le développement de produits critiques Convient moins aux produits peu critiques 	<ul style="list-style-type: none"> Non testé sur des systèmes critiques Difficultés potentielles dues au simple design et au manque de documentation
Dynamisme (% de changement d'exigences/mois)	<ul style="list-style-type: none"> Plans détaillés et design précis sont bien adaptés aux environnements stables Mais source de travail importante quand l'environnement est changeant 	<ul style="list-style-type: none"> Simple design & refactoring continu sont bien adaptés aux environnements forts changeants Mais peut être source de travail supplémentaire dans un environnement stable
Personnel (expérience)	<ul style="list-style-type: none"> Demande du personnel expérimenté durant la définition du projet Utilisation de juniors par la suite 	<ul style="list-style-type: none"> Demande la présence d'une masse critique de personne expérimentées Risqué avec des juniors
Culture	<ul style="list-style-type: none"> Organisations où les rôles sont clairement définis dans des procédures reconnues 	<ul style="list-style-type: none"> Adapté aux organisations laissant une grande marge de liberté aux employés