Les méthodes agiles

Les méthodes agiles sont des groupes de pratiques de pilotage et de réalisation de projets. Elles ont pour origine le manifeste Agile, rédigé en 2001, qui consacre le terme d'« agile » pour référencer de multiples méthodes existantes.

1. Historique

Les méthodes agiles se veulent plus pragmatiques que les méthodes traditionnelles de développement logiciel, impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes. Elles reposent sur un cycle de développement itératif, incrémental et adaptatif et doivent respecter quatre valeurs fondamentales déclinées en douze principes desquels découlent une base de pratiques, soit communes, soit complémentaires.

Les méthodes pouvant être qualifiées d'agiles, depuis la publication en 1991 du manifeste Agile, sont le RAD (développement rapide d'applications). Les deux méthodes agiles les plus utilisées sont la méthode Scrum qui fut publiée en 2001 ainsi que la méthode XP Extreme programming publiée en 1999. Ces deux méthodes sont d'ailleurs techniquement complémentaires dans le cas d'un projet de développement de SI, XP proposant alors les techniques d'obtention de la qualité du code.

- L'apport méthodologique d'XP réside dans la préconisation de pousser à l'extrême les principales pratiques de qualité de la construction applicative ainsi que les techniques adaptatives d'estimation consensuelle, de planification pilotée par l'utilisateur et de reporting visuel en temps réel de l'avancement ainsi que des problèmes rencontrés (affichage mural à base de post-it).
- Scrum propose un ensemble réduit de pratiques concentrées sur le développement de l'adaptabilité par l'apprentissage et l'auto-organisation.

2. Fondements

Les méthodes agiles prônent 4 valeurs fondamentales :

1. Individus et interactions plutôt que processus et outils
2. Fonctionnalités opérationnelles plutôt que documentation exhaustive
3. Collaboration avec le client plutôt que contractualisation des relations
4. Acceptation du changement plutôt que conformité aux plans

et douze principes généraux :

1. Satisfaire le client en priorité
2. Accueillir favorablement les demandes de changement
3. Livrer le plus souvent possible des versions opérationnelles de l'application
4. Assurer une coopération permanente entre le client et l'équipe projet
5. Construire des projets autour d'individus motivés
6. Privilégier la conversation en face à face
7. Mesurer l'avancement du projet en termes de fonctionnalités de l'application
8. Faire avancer le projet à un rythme soutenable et constant
9. Porter une attention continue à l'excellence technique et à la conception
10. Faire simple
11. Responsabiliser les équipes
12. Ajuster à intervalles réguliers son comportement et ses processus pour être plus efficace

3. L'extreme programming

3.1. Origine

L'eXtreme Programming (XP[i]) a été inventée par Kent Beck, Ward Cunningham, Ron Jeffries et Palleja Xavier pendant leur travail sur un projet de calcul des rémunérations chez Chrysler. Kent Beck, chef de projet en mars 1996, commença à affiner la méthode de développement utilisée sur le projet. Celle-ci est née officiellement en octobre 1999 avec le livre Extreme Programming Explained de Kent Beck.

3.2. Pratiques extrêmes

En génie logiciel, eXtreme Programming (XP) est une méthode agile plus particulièrement orientée sur l'aspect réalisation d'une application, sans pour autant négliger l'aspect gestion de projet. XP est adapté aux équipes réduites avec des besoins changeants. XP pousse à l'extrême des principes simples.

Dans le livre Extreme Programming Explained, la méthode est définie comme :

- une tentative de réconcilier l'humain avec la productivité ;
- un mécanisme pour faciliter le changement social ;
- une voie d'amélioration ;
- un style de développement ;
- une discipline de développement d'applications informatiques.

Son but principal est de réduire les coûts du changement. Dans les méthodes traditionnelles, les besoins sont définis et souvent fixés au départ du projet informatique ce qui accroît les coûts ultérieurs de modifications. XP s'attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques.

Les principes de cette méthode ne sont pas nouveaux : ils existent dans l'industrie du logiciel depuis des dizaines d'années et dans les méthodes de management depuis encore plus longtemps. L'originalité de la méthode est de les pousser à l'extrême puisque :

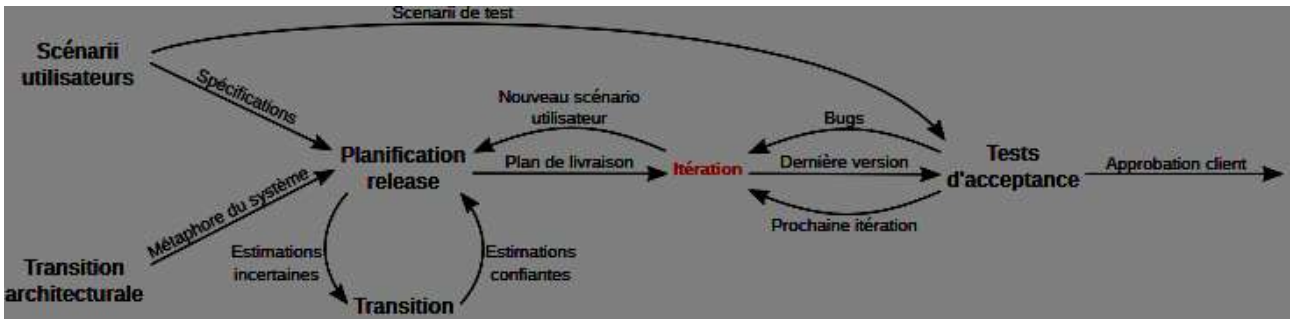
- la revue de code est une bonne pratique, elle sera faite en permanence (par un binôme)
- les tests sont utiles, ils seront faits systématiquement avant chaque mise en œuvre
- la conception est importante, elle sera faite tout au long du projet (refactoring)
- la simplicité permet d'avancer plus vite, nous choisirons toujours la solution la plus simple
- la compréhension est importante, nous définirons et ferons évoluer ensemble des métaphores
- l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour
- les besoins évoluent vite, nous ferons des cycles de développement très rapides pour nous adapter au changement

3.3. Cycle de développement

L'extreme programming repose sur des cycles rapides de développement (des itérations de quelques semaines) dont les étapes sont les suivantes :

- une phase d'exploration détermine les scénarios « client » à fournir pendant cette itération
- l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels
- chaque développeur s'attribue des tâches et les réalise avec un binôme

- lorsque tous les tests fonctionnels passent, le produit est livré



Le cycle se répète tant que le client peut fournir des scénarios à livrer. Généralement le cycle de la première livraison se caractérise par sa durée et le volume important de fonctionnalités embarquées. Après la première mise en production, les itérations peuvent devenir plus courtes (une semaine par exemple).

3.4. Programmation comme discipline collective

Tout en mettant l'accent sur les bonnes pratiques de programmation, XP préconise un déroulement par itérations courtes et gérées collectivement, avec une implication constante du client. Il en découle une redéfinition de la relation entre client et fournisseur avec des résultats surprenants sur le plan de la qualité de code, de délais et de satisfaction de la demande du client.

3.4.1. Valeurs

L'extreme programming repose sur cinq valeurs fondamentales :

1. Communication : C'est le moyen fondamental pour éviter les problèmes. Les pratiques que préconise l'XP imposent une communication intense. Les tests, la programmation en binôme et le jeu du planning obligent les développeurs, les décideurs et les clients à communiquer. Si un manque apparaît malgré tout, un coach se charge de l'identifier et de remettre ces personnes en contact.
2. Simplicité : La façon la plus simple d'arriver au résultat est la meilleure. Anticiper les extensions futures est une perte de temps. Une application simple sera plus facile à faire évoluer.
3. Feedback : Le retour d'information est primordial pour le programmeur et le client. Les tests unitaires indiquent si le code fonctionne. Les tests fonctionnels donnent l'avancement du projet. Les livraisons fréquentes permettent de tester les fonctionnalités rapidement.
4. Courage : Certains changements demandent beaucoup de courage. Il faut parfois changer l'architecture d'un projet, jeter du code pour en produire un meilleur ou essayer une nouvelle technique. Le courage permet de sortir d'une situation inadaptée. C'est difficile, mais la simplicité, le feedback et la communication rendent ces tâches accessibles.
5. Respect : Cette valeur fut ajoutée dans la deuxième édition de Extreme Programming Explained de K. Beck. Cette valeur inclut le respect pour les autres, ainsi que le respect de soi. Les programmeurs ne devraient jamais valider les modifications qui cassent la compilation, qui font échouer les tests unitaires existants ou qui retardent le travail de leurs pairs. Les membres respectent leur propre travail en cherchant toujours la qualité et la meilleure conception pour la solution et cela grâce au refactoring.

3.4.2. Pratiques

Ces cinq valeurs se déclinent en treize pratiques qui se renforcent mutuellement :

1. Client sur site : Un représentant du client doit, si possible, être présent pendant toute la durée

du projet. Il doit avoir les connaissances de l'utilisateur final et avoir une vision globale du résultat à obtenir. Il réalise son travail habituel tout en étant disponible pour répondre aux questions de l'équipe.

2. Jeu du planning ou planning poker : Le client crée des scénarios pour les fonctionnalités qu'il souhaite obtenir. L'équipe évalue le temps nécessaire pour les mettre en œuvre. Le client sélectionne ensuite les scénarios en fonction des priorités et du temps disponible.
3. Intégration continue : Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.
4. Petites livraisons : Les livraisons doivent être les plus fréquentes possible. L'intégration continue et les tests réduisent considérablement le coût de livraison.
5. Rythme soutenable : L'équipe ne fait pas d'heures supplémentaires. Si le cas se présente, il faut revoir le planning. Un développeur fatigué travaille mal.
6. Tests de recette (ou tests fonctionnels) : À partir des scénarios définis par le client, l'équipe crée des procédures de test qui permettent de vérifier l'avancement du développement. Lorsque tous les tests fonctionnels passent, l'itération est terminée. Ces tests sont souvent automatisés mais ce n'est pas toujours possible.

En effet, seuls les tests de non régression peuvent être potentiellement automatisés du fait de leur récurrence.

La recette fonctionnelle d'une application est de plus en plus souvent confiée à des experts du test indépendants des développeurs.

7. Tests unitaires : Avant de mettre en œuvre une fonctionnalité, le développeur écrit un test qui vérifiera que son programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, on lance tous les tests écrits par tous les développeurs, et on sait immédiatement si quelque chose ne fonctionne plus.
8. Conception simple : L'objectif d'une itération est de mettre en œuvre les scénarios sélectionnés par le client et uniquement cela. Envisager les prochaines évolutions ferait perdre du temps sans avoir la garantie d'un gain ultérieur. Les tests permettront de changer l'architecture plus tard si nécessaire. Plus l'application est simple, plus il sera facile de la faire évoluer lors des prochaines itérations.
9. Utilisation de métaphores : On utilise des métaphores et des analogies pour décrire le système et son fonctionnement. Le fonctionnel et le technique se comprennent beaucoup mieux lorsqu'ils sont d'accord sur les termes qu'ils emploient.
10. Refactoring (ou remaniement du code) : Amélioration régulière de la qualité du code sans en modifier le comportement. On retravaille le code pour repartir sur de meilleures bases tout en gardant les mêmes fonctionnalités. Les phases de refactoring n'apportent rien au client mais permettent aux développeurs d'avancer dans de meilleures conditions et donc plus vite.
11. Appropriation collective du code : L'équipe est collectivement responsable de l'application. Chaque développeur peut faire des modifications dans toutes les portions du code, même celles qu'il n'a pas écrites. Les tests diront si quelque chose ne fonctionne plus.
12. Convention de nommage : Puisque tous les développeurs interviennent sur tout le code, il est indispensable d'établir et de respecter des normes de nommage pour les variables, méthodes, objets, classes, fichiers, etc.
13. Programmation en binôme : La programmation se fait par deux. Le premier appelé driver (ou pilote) tient le clavier. C'est lui qui va travailler sur la portion de code à écrire. Le

second appelé Partner (ou copilote) est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes. Les développeurs changent fréquemment de partenaire ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

3.4.3. Autres principes

- Ne pas ajouter de fonctionnalités plus tôt que prévu
- N'optimiser qu'à la toute fin

Cette méthode s'appuie sur :

- une forte réactivité au changement des besoins du client ;
- un travail d'équipe ;
- la qualité du code ;
- la qualité des tests effectués au plus tôt.

4. Scrum

4.1. Caractéristiques

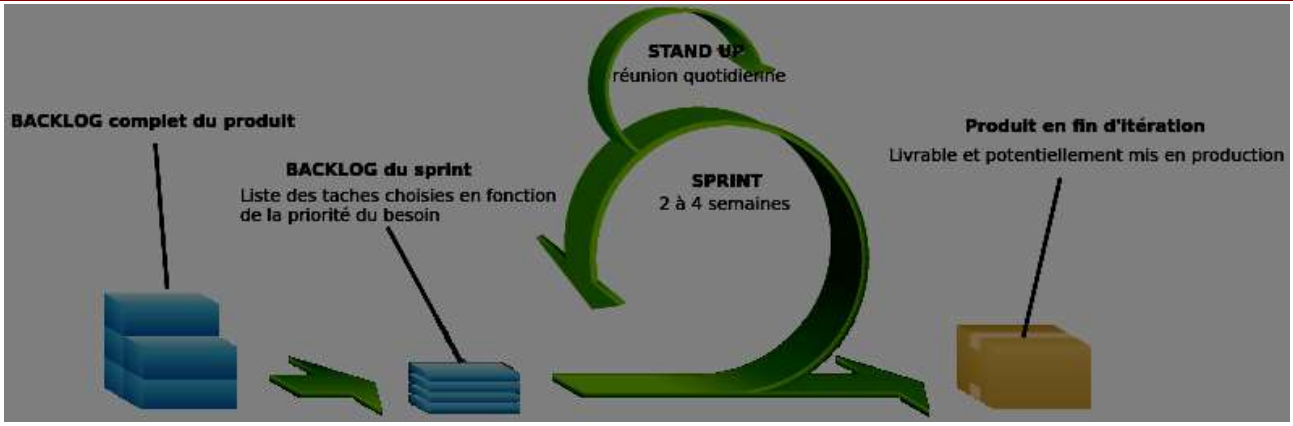
Scrum est un schéma d'organisation de développement de produits complexes. Il est défini par ses créateurs comme un « cadre de travail holistique itératif qui se concentre sur les buts communs en livrant de manière productive et créative des produits de la plus grande valeur possible ».

Le framework s'appuie sur le découpage d'un projet en boîtes de temps, nommées « sprints ». Les sprints peuvent durer entre quelques heures et un mois (avec une préférence pour deux semaines). Chaque sprint commence par une estimation suivie d'une planification opérationnelle. Le sprint se termine par une démonstration de ce qui a été achevé. Avant de démarrer un nouveau sprint, l'équipe réalise une rétrospective. Cette technique analyse le déroulement du sprint achevé, afin d'améliorer ses pratiques. Le flot de travail de l'équipe de développement est facilité par son auto-organisation, il n'y aura donc pas de gestionnaire de projet.

La création de framework de développement logiciel hybride couplant Scrum et d'autres frameworks est commune puisque Scrum ne couvre pas le cycle de développement de produit. Par exemple, on pourra utiliser des pratiques issues de l'extreme programming, de la phase de construction structurée de la méthode RAD, ou un ensemble de pratiques de qualité du logiciel émergeant du vécu de l'équipe projet.

4.2. Les trois piliers de scrum

La méthode scrum est fondée sur la conviction que le développement logiciel est une activité par nature non-déterministe et que l'ensemble des activités de réalisation d'un projet complexe ne peut être anticipé et planifié. C'est en cela que le scrum s'oppose aux démarches prédictives telles que le cycle en V. Pour répondre à cette imprédictibilité, la méthodologie scrum propose un modèle de contrôle de processus fondée sur l'empirisme, via l'adaptation continue aux conditions réelles de l'activité et une réaction rapide aux changements. L'analyse des conditions réelles d'activité lors des rétrospectives de fin de Sprint et le plan d'amélioration continue qui en découle sont réalisés à intervalle de temps régulier, donnant lieu à un cycle de développement incrémental (Sprint).



une itération selon la méthode scrum

La méthode scrum a été conçue lors de projets de développement de logiciels. Elle peut aussi être utilisée par des équipes de maintenance. La méthode scrum peut théoriquement s'appliquer à n'importe quel contexte ou à un groupe de personnes qui travaillent ensemble pour atteindre un but commun.

Un principe fort des méthodes Agiles est la participation active du client. Cela permet de choisir plus finement les fonctionnalités réalisées à chaque incrément. Avant le démarrage du sprint 1, les objectifs sont définis lors d'un sprint 0. La mêlée (scrum meeting) a lieu quotidiennement et des réunions spécifiques permettent de lever les obstacles bloquants. Le Sprint a une durée variable (idéalement deux semaines). Après chaque sprint, une démonstration suivie d'une rétrospective ont lieu. Le propriétaire du produit peut à tout moment compléter ou modifier la liste des fonctionnalités à produire pour les prochains sprints. Sans modifier le but du sprint en cours, celui-ci peut être affiné et faire l'objet d'une renégociation entre le propriétaire du produit et l'équipe de développement à la suite de nouvelles connaissances. Si le but du sprint devient obsolète, le propriétaire du produit a la capacité d'annuler un sprint en cours.

Chaque sprint constitue donc un incrément, facilitant le pilotage du projet. La notion d'itération couvre l'adaptabilité au quotidien. Cette adaptabilité est limitée par le but immuable d'un sprint.

Scrum est un processus empirique : il se base sur l'expérience du terrain. Il s'appuie sur trois piliers :

1. Transparence : Scrum met l'accent sur le fait d'avoir un langage commun entre l'équipe et le management. Ce langage commun doit permettre à tout observateur d'obtenir rapidement une bonne compréhension du projet.
2. Inspection : À intervalle régulier, Scrum propose de faire le point sur les différents artéfacts produits, afin de détecter toute variation indésirable.

Ces inspections ne doivent pas être faites trop fréquemment, ou par un inspecteur mal formé : cela nuirait à l'avancement du projet.

3. Adaptation : Si une dérive est constatée pendant l'inspection, le processus doit alors être adapté. Scrum fournit des rituels, durant lesquels cette adaptation est possible. Il s'agit de la réunion de planification de sprint, de la mêlée quotidienne, de la revue de sprint ainsi que de la rétrospective de sprint.

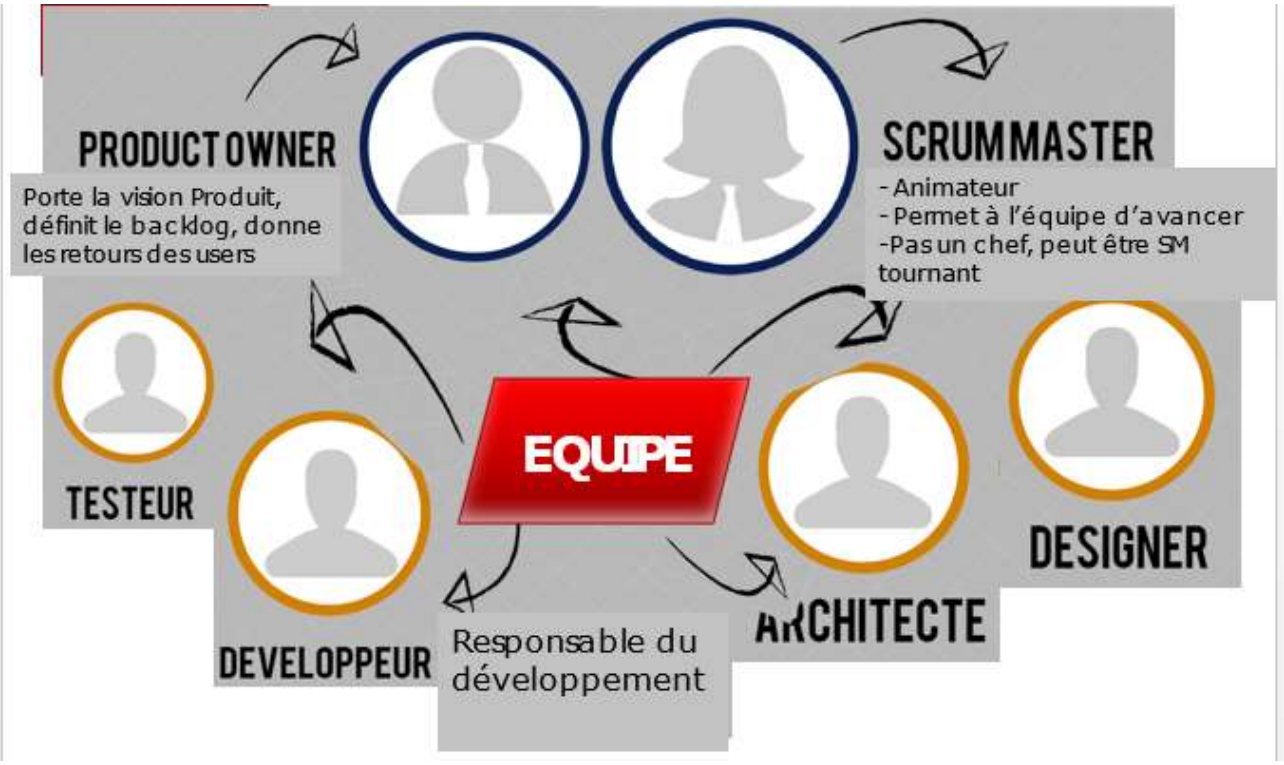
4.3. Rôles, événements et artefacts

Le cadre du scrum consiste en la définition des rôles projets, activités ou artefacts, et réunions ou événements.

4.3.1. Rôles

Scrum définit trois rôles : le propriétaire du produit (product owner), le scrum master et l'équipe de

développement. Il est à noter que le rôle de l'équipe développeur couvre plusieurs métiers d'une organisation traditionnelle.



1. **Propriétaire du produit** : Le propriétaire du produit (product owner) est le représentant des clients et des utilisateurs. Il est « responsable de maximiser la valeur du produit et du travail de l'équipe de développement ». Il s'agit d'« une personne et non d'un comité ». Il est seul à diriger l'activité de l'équipe de développement à qui il n'est « pas permis de suivre les instructions d'une autre personne. »

   De ce fait, cet acteur se charge de différents rôles et responsabilités :

   - Il explicite les éléments (items) du carnet du produit.
   - Il définit l'ordre dans lequel les fonctionnalités seront développées et prend les décisions importantes concernant l'orientation du projet.
   - Il s'assure que le carnet du produit est visible et compris de l'équipe de développement.
   - Il participe avec l'équipe à la définition de l'objectif du Sprint au début de celui-ci (Sprint Planning). L'objectif peut être fonctionnel, technique ou organisationnel. Si l'objectif devient obsolète pendant le sprint, il a alors la possibilité d'interrompre le sprint en cours.

   Dans l'idéal, le propriétaire du produit travaille dans la même pièce que l'équipe. Il est important qu'il reste très disponible pour répondre aux questions de l'équipe et pour lui donner son avis sur divers aspects du logiciel.

2. **Maître de mêlée** : Le maître de mêlée (scrum master) est responsable de la compréhension, de l'adhésion et de la mise en œuvre du framework. C'est un « leader au service de l'équipe », il assiste chaque rôle de l'équipe scrum dans son activité et promeut le changement des interactions entre les rôles dans le but de maximiser la valeur de ce que produit l'équipe. Son autorité s'exerce sur le processus de développement (définition de la durée des Sprints, des modalités de tenues et de l'ordre du jour des réunions scrum…), mais il ne dispose d'aucune autorité sur les autres membres de l'équipe scrum.

Ce n'est pas un chef de projet, ni un développeur, ni un intermédiaire de communication avec les clients. Le rôle de scrum master ne peut pas être cumulé avec celui de propriétaire du produit. Les attributions du « chef de projet » présent dans d'autres approches sont distribuées dans les différents rôles de l'équipe scrum. L'existence du rôle de chef de projet dans un contexte scrum est le signe d'une « méconnaissance fondamentale de scrum » et est perçue comme contre-productive et menant à de « piètres résultats ».

En tant que facilitateur, il aide l'équipe à déterminer quelles interactions avec l'extérieur lui sont utiles, et lesquelles sont freinantes. Il aide alors à maximiser la valeur produite par l'équipe.

Parmi ses attributions :

- communiquer la vision et les objectifs à l'équipe
- apprendre au propriétaire du produit à rédiger les composantes du carnet du produit
- faciliter les rituels de scrum
- coacher l'équipe de développement
- faciliter son intégration à l'entreprise, surtout si celle-ci n'est pas pleinement agile
- écarter les éléments pouvant perturber l'équipe
- aider l'adoption de la culture agile au niveau de l'entreprise
- travailler avec les autres Facilitateurs/animateurs pour coordonner plusieurs équipes

3. **Équipe de développement** : L'équipe de développement est constituée de 3 à 9 personnes et a pour responsabilité de livrer à chaque (sprint) fin d'itération une nouvelle version de l'application enrichie de nouvelles fonctionnalités et respectant le niveau de qualité nécessaire pour être livrée.

   L'équipe ne comporte pas de rôles prédéfinis ; elle est « structurée et habilitée par l'entreprise à organiser et gérer son propre travail ».

   Elle est auto-organisée et choisit la façon d'accomplir son travail, sans que ce soit imposé par une personne externe. Il n'y a pas non plus de notion de hiérarchie interne : toutes les décisions sont prises ensemble. Ce mode d'organisation a pour objectif d'augmenter l'efficacité de travail de l'équipe.

   Elle est pluridisciplinaire et comporte toutes les compétences pour réaliser son projet, sans faire appel à des personnes externes à celle-ci.

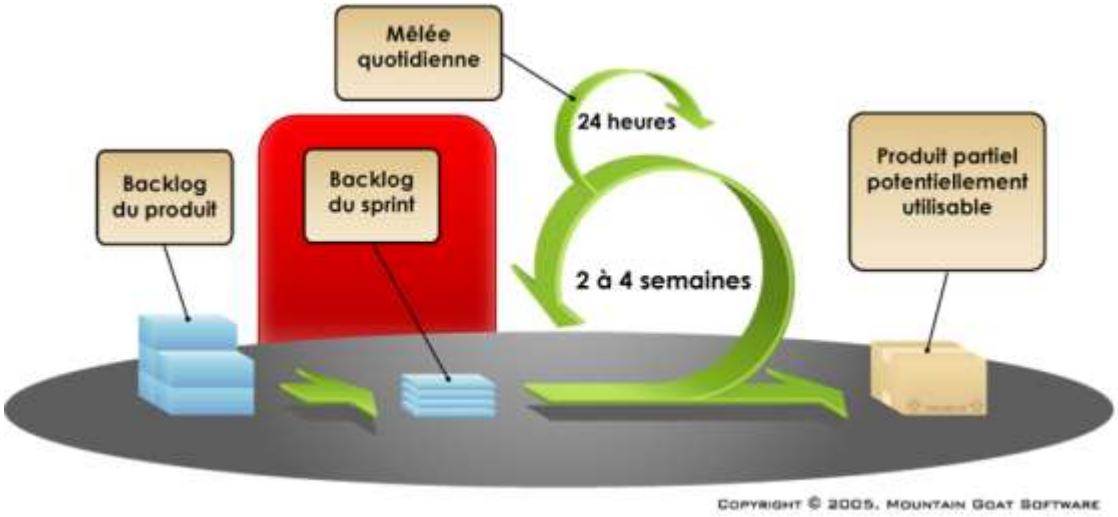
   L'objectif de l'équipe est de livrer le produit par petits incréments. Ainsi, à tout instant, il existe une version du produit « potentiellement utilisable » disponible.

   L'équipe s'adresse directement au propriétaire du produit, et ne prend ses instructions que de lui. Son activité est issue du carnet de produit uniquement. Elle ne peut pas être multi-produits.

4.3.2. Événements

Toutes les activités (sprint, réunions de planning, revues, rétrospectives et mêlées) décrites dans le framework scrum sont effectuées lors de chaque itération dans un temps.

- **Sprint** : Le sprint est une période d'un mois au maximum, au bout de laquelle l'équipe délivre un incrément du produit, potentiellement livrable. Une fois la durée choisie, elle reste constante pendant toute la durée du développement. Un nouveau sprint démarre dès la fin du précédent.



Chaque sprint possède un but et on lui associe une liste d'éléments du carnet du produit (fonctionnalités) à réaliser.

Durant un sprint :

- l'objet du sprint ne peut être modifié
- la composition de l'équipe reste constante
- la qualité n'est pas négociable
- la liste d'éléments est sujette à négociations entre le propriétaire du produit et l'équipe de développement

La limitation est d'un mois afin de limiter la complexité et donc les risques liés au sprint.

Si l'objet du sprint devient obsolète pendant celui-ci, le propriétaire du produit peut décider de l'annuler. Dans ce cas, les développements terminés sont revus par le propriétaire du produit, qui peut décider de les accepter. Les éléments du sprint n'étant pas acceptés sont ré-estimés et remis dans le carnet du produit. Un nouveau sprint démarre alors.

- **Mêlée quotidienne** : La mêlée quotidienne (daily scrum) est une réunion de planification « juste à temps » et permet aux développeurs de faire un point de coordination sur les tâches en cours et sur les difficultés rencontrées. Cette réunion dure 15 minutes au maximum. Seule l'équipe de développement intervient. Le Scrum Master peut intervenir comme facilitateur en début de mise en place de Scrum. Toute autre personne peut assister mais ne peut pas intervenir.

  À tour de rôle, chaque membre aborde trois sujets :

  - ce qu'il a réalisé la veille
  - ce qu'il compte réaliser aujourd'hui pour atteindre l'objet du sprint
  - les obstacles qui empêchent l'équipe d'atteindre le but du sprint

Si le besoin s'en fait sentir, des discussions sont alors menées librement après la clôture de la mêlée pour traiter des sujets levés en séance.

Cette réunion permet la synchronisation de l'équipe, l'évaluation de l'avancement vers l'objectif de l'itération, la collecte d'information nécessaire à l'auto-organisation. C'est le niveau quotidien des principes inspection et adaptation de scrum.

- Réunion de planification d'un Sprint : Toute l'équipe scrum est présente à cette réunion, qui ne doit pas durer plus de 8 heures pour un sprint d'un mois. Pour un sprint plus court, la durée est en général réduite mais peut durer 8h. À l'issue de cette réunion, l'équipe a décidé des éléments du carnet du produit qu'elle traitera dans le cadre de la prochaine itération, et comment elle

s'organisera pour y parvenir.

La réunion de planification de sprint (sprint planning meeting) se passe en deux temps. Dans la première partie, l'équipe de développement cherche à prévoir ce qui sera développé durant le prochain sprint. À l'entrée de cette phase, l'équipe doit avoir à sa disposition :

- le carnet du produit priorisé
- la capacité de production (vélocité) prévue pour ce sprint (estimation théorique pour le premier sprint et basée sur la productivité réelle obtenue pour les suivants)

L'équipe et le propriétaire du produit détermine le but du sprint. Dans un second temps, l'équipe, aidée du propriétaire du produit, se focalise sur la manière dont ses membres atteindront le but du sprint, en découpant en activités d'une durée d'une journée au plus le travail à effectuer pendant le sprint.

- Revue de sprint : À la fin du sprint, l'équipe scrum et les parties-prenantes invitées se réunissent pour effectuer la revue de sprint, qui dure au maximum quatre heures. L'objectif de la revue de sprint est de valider l'incrément de produit qui a été réalisé pendant le sprint. L'équipe énonce les éléments du carnet de produit sélectionnés en début de sprint. L'équipe présente les éléments finis (complètement réalisés). Les éléments non finis (partiellement réalisés) ne sont pas présentés.

Une fois le bilan du sprint réalisé, l'équipe de développement et le propriétaire du produit mettent à jour le carnet du produit en fonction de ce qui a été réalisé (fini). Ils discutent avec les parties-prenantes de l'état courant du projet (budget, financement, conditions du marché), pour ajuster les éléments de carnet de produit et la planification selon les opportunités découvertes.

- Rétrospective du sprint : La rétrospective du sprint est faite en interne à l'équipe scrum (équipe de réalisation, propriétaire du produit et scrum master). Elle dure trois heures pour un sprint d'un mois. Elle a pour but l'adaptation aux changements qui surviennent au cours du projet et l'amélioration continue du processus de réalisation.

L'objectif est d'inspecter l'itération précédente, afin de déterminer quels sont les éléments du processus de développement qui ont bien fonctionné et ceux qui sont à améliorer. L'équipe de développement déduit un plan d'actions d'amélioration qu'elle mettra en place lors de l'itération suivante.

4.3.3. Artefacts

- **Carnet du produit** (product backlog) : Le carnet de produit est « une liste ordonnée de tout ce qui pourrait être requis dans le produit et est l'unique source des besoins pour tous les changements à effectuer sur le produit »10. C'est un document qui évolue constamment au cours de la vie du produit et n'est « jamais fini ».

  Chaque élément du carnet représente une fonctionnalité, besoin, amélioration et correctif, auquel sont associés une description, une estimation de l'effort nécessaire à la réalisation de l'élément et une grandeur permettant d'ordonner les éléments entre eux. Le carnet de produit, son évolution et sa publication sont de la responsabilité du propriétaire du produit. Il peut changer à discrétion l'ordre des éléments, en ajouter, modifier le découpage en éléments, modifier leur description, ou supprimer des éléments qui n'ont pas encore été réalisés par l'équipe de développement. Les éléments en tête du carnet de produit sont destinés à être traités dans la prochaine itération et sont les plus finement décrits et estimés. Ils sont dits « prêts » dans la terminologie scrum. Les éléments moins prioritaires peuvent être découpés plus grossièrement en attendant de devenir prioritaires et affinés à leur tour.

  L'activité d'affinage du carnet de produit et de ses éléments est effectuée conjointement par le propriétaire du produit et par l'équipe de réalisation. C'est notamment elle seule qui a le mot final sur les estimations des éléments du carnet du produit.

- **Carnet de sprint** (sprint backlog) : En début de sprint, un but est décidé. Pour atteindre cet objectif, l'équipe de développement choisit lors de la réunion de planification de sprint quels éléments du carnet de produit seront réalisés. Ces éléments sont alors groupés dans un carnet de sprint.

  Chaque équipe met à jour régulièrement le carnet de sprint au cours de son activité, afin que celui-ci donne une vision la plus précise possible de ce que l'équipe prévoit de réaliser pour atteindre l'objectif du sprint. Le carnet de sprint est sous la responsabilité de l'équipe et elle est seule à pouvoir le modifier en cours d'itération.

- **Incrément de produit** : L'incrément de produit est l'ensemble des éléments du carnet de produit finis pendant ce sprint, et aussi ceux finis pendant les sprints précédents. Les éléments sont finis lorsqu'ils remplissent la définition de fini. Cet incrément doit être utilisable, qu'il soit publié ou non.

---

i