L'entreprise doit avoir l'agilité du banc de poisson qui agit et réagit avec vitesse, simultanéité et précision.[1]

Méthodes traditionnelles ou méthodes agiles?

Depuis des décennies, les projets informatiques sont gérés avec une **approche classique**, basée sur des **activités séquentielles** : on recueille les besoins, on définit le produit, on le développe, on l'implémente puis on le teste avant de le livrer au client. Ces méthodes classiques sont souvent qualifiées par le vocable « ***Waterfall*** » (« *En cascade* »).

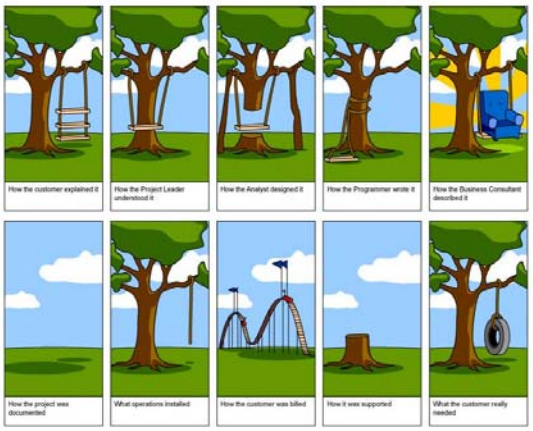
Nous avons vu, lors du premier cours, que cette méthode classique n'est pas toujours gage de succès. Rappelons ici, les statistiques du Standish Group :

- **moins de 30%** des projets informatiques sont qualifiés de **succès**, i.-e qu'ils sont livrés à temps, à l'intérieur des budgets et des spécifications originales et qu'à l'inverse plus de 70%
- **plus de la moitié des projets** (53%) sont qualifiés d'**échec partiel** : ils sont livrés et opérationnels, mais ont moins de fonctions que prévues, dépassent les budgets et/ou les échéanciers;
- enfin, **près de 20%** des projets sont des **échecs complets**, c'est-à-dire abandonnés en cours de route ou que les résultats livrés ne sont jamais utilisés.

Les principales sources d'échec sont les suivantes :

- Manque d'implication des utilisateurs;
- Imprécision du cahier des charges;
- Absence de priorisation ou de valorisation des besoins exprimés par le client;
- Faible implication du client;
- Changements aux exigences et spécifications en cours de route;
- Manque de rigueur dans le suivi du projet;
- Démotivation ou le manque d'implication des membres de l'équipe de projet;
- Absence de flux de communication entre les membres de l'équipe de projet;
- Inadéquation ou l'incompétence des ressources;
- Contraintes de délais trop courts
- Etc.

[1] BARRAND, Jérôme, *Le manager agile*, Dunod, 2006.



Pas évident de réussir un projet si au point de départ, le client a de la difficulté à exprimer ce qu'il veut!!! Et qu'en plus, il y a des marges d'interprétation qui induisent un écart souvent significatif entre ce que le client a imaginé et ce que l'intégrateur a compris....

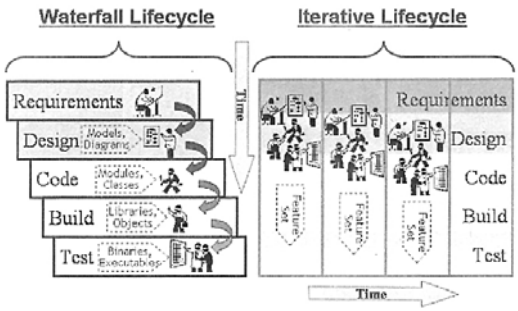
Une alternative : les méthodes agiles

Qu'est-ce qu'une méthode agile?

Une méthode agile est une **approche itérative et incrémentale**, qui est menée dans un esprit collaboratif, avec juste ce qu'il faut de formalisme. Elle génère un produit de haute qualité tout en prenant en compte l'évolution des besoins des clients.

Le **principe du développement itératif** consiste à découper le projet en plusieurs étapes d'une durée fixe (« **timebox** ») de une à 4 semaines, appelées « **sprints** ». Au cours d'un « **sprint** », une **version minimale** du produit attendu est développée puis soumise, dans sa version intermédiaire au client pour validation. Ainsi, un sprint aboutit toujours à la livraison d'un produit **partiel mais fonctionnel.** Les fonctionnalités sont ainsi intégrées au fur et à mesure du cycle de vie sur un **mode incrémental**, le système s'enrichissant progressivement pour atteindre les niveaux de satisfaction et de qualité requis.

Chaque **itération est un mini-projet** qui comporte **toutes les activités de développement**, **menées en parallèle**: analyse, conception, codage, test, etc., sans oublier les activités de gestion de projet. L'objectif est d'obtenir, au terme de chaque itération, un **sous-ensemble opérationnel du système** cible et, au terme de la dernière itération, la version finale du produit.



Origine et valeurs des méthodes agiles

Le mouvement des méthodes agiles est né en 2001 aux États-Unis. Devant l'observation faite du taux important d'échec des projets informatiques, notamment dans les années 1990, **dix-sept experts** en développement logiciel, qui avaient chacun déjà mis au point et expérimenté de nouvelles méthodes, se sont réunis afin d'échanger et de trouver un socle commun de valeurs et de bonnes pratiques qu'ils considéraient être des facteurs clés de succès dans le développement logiciel.

Le résultat de cette réflexion a abouti au **Manifeste pour le développement agile**[2] et la création de l'**Agile Alliance**[3], chargée de promouvoir l'agilité dans les organisations et d'apporter du soutien aux équipes qui veulent démarrer un projet agile.

Le manifeste **décline quatre valeurs en treize principes** applicables dans toute démarche agile. Chaque méthode adopte ensuite sa propre terminologie et préconise un certain nombre de pratiques.

Quelles sont les quatre valeurs du Manifeste?

- Les individus et leurs interactions avant les processus et les outils;
- Des fonctionnalités opérationnelles avant la documentation;
- Collaboration avec le client plutôt que contractualisation des relations;
- Acceptation du changement plutôt que conformité aux plans.

Cela ne signifie évidemment pas qu'aucun processus n'est défini, qu'on ne se dote d'aucun outil; bien entendu, des plans et une documentation, certes plus réduits sont produits.

Quels sont les treize principes du Manifeste?

Principe	Description
Notre **priorité** est de **satisfaire le client** en lui livrant très tôt et régulièrement des versions opérationnelles de l'application, source de valeur.	Grâce au développement itératif, chaque livraison intermédiaire donne lieu à une validation par le client; son feedback est essentiel pour garantir la conformité de la livraison avec ses attentes, pour prendre en compte ses remarques et (re)prioriser
Accepter le changement dans les exigences, même tard dans le cycle de vie, pour garantir la compétitivité du client.	Cet état d'esprit caractérise une équipe agile qui démontre ainsi sa capacité à comprendre et apprendre comment satisfaire encore mieux la demande.
Livrer le plus souvent possible des versions opérationnelles de l'application, à une **fréquence allant de deux semaines à deux mois**.	Une version intermédiaire du produit final, visible et testable, satisfait davantage le client qu'une documentation à valider. Il a, de ce fait, la preuve tangible que le projet avance et peut exprimer son point de vue sur le résultat présenté.

[2] http://agilemanifesto.org/
[3] http://agilealliance.org/

Principe	Description
Client et **développeurs** doivent **coopérer quotidiennement** tout au long du projet.	Les relations conflictuelles ne font pas partie de l'esprit agile; on préférera des relations collaboratives et de partenariat basées sur la confiance et le consensus. Le client (ou son représentant) est accessible et disponible, totalement impliqué dans toutes les phases du projet.
Construire des projets autour d'individus motivés. Leur donner l'environnement et le support dont ils ont besoin et leur faire confiance pour remplir leur mission.	Le facteur clé du succès est l'équipe. Tout obstacle à son bon fonctionnement devra être levé; un changement, s'il s'avère nécessaire, sera apporté aux processus, aux outils, à l'environnement, à la composition de l'équipe.
La **méthode la plus efficace de communiquer** des informations à une équipe et au sein de celles-ci reste la **conversation en face à face**.	Par défaut, on privilégie l'échange oral à l'écrit, pour lever toute ambiguïté et favoriser la rapidité de la compréhension. Tout ne peut être formalisé par écrit, notamment la « connaissance tacite », c'est-à-dire l'information « informelle », la culture du projet, détenues par chacun au sein d'une équipe.
Le **fonctionnement de l'application** est le **premier indicateur d'avancement** du projet	Il n'existe pas d'autre indicateur plus pertinent que le pourcentage ou le nombre d'exigences satisfaites; on ne mesure pas un projet à la quantité de documents produits ou au nombre de lignes de code, non significatifs pour le client.
Les méthodes agiles recommandent que le **projet avance à un rythme soutenable**.	La qualité du travail fourni dépend du rythme de travail qui doit être adapté en fonction des spécificités du projet. Le **rythme** doit être **soutenu** et **soutenable** sur la durée du projet.
Sponsors, développeurs et utilisateurs devraient pouvoir **maintenir un rythme constant indéfiniment**.	Ce rythme est à déterminer par l'ensemble des membres de l'équipe et par le client, en fonction de la productivité de l'équipe et des priorités du client. Mais travailler en heures supplémentaires, surtout pour corriger des bogues ou des régressions, n'apporte aucune valeur ajoutés.
Porter une **attention continue à l'excellence technique** et à la conception de l'agilité.	Maintenir un code propre, évolutif et performant est un objectif permanent de l'équipe; il ne s'agit pas de produire rapidement du code non exploitable par les autres, ni du « jetable ». De plus, cela évite surtout d'enliser les développements ultérieurs, avec des modifications cassant un développement fragile, nécessitant des interventions à des endroits variés du code.
La **simplicité** – art de maximiser la quantité de travail non fait – est essentielle.	La simplicité garantit l'évolutivité du système. La complexité, au contraire, coûte davantage et rend plus difficiles les évolutions inhérentes au développement incrémental. La conception ne doit comporter que des éléments utiles.
Les meilleures architectures, spécifications et conceptions sont le fruit d'**équipes qui s'auto-organisent**.	Le chef de projet agile n'est plus celui qui attribue des tâches. L'équipe, elle-même, se responsabilise et définit ses travaux à réaliser, le partage des tâches s'effectuant sur la base du volontariat.
À **intervalles réguliers**, l'ensemble de **l'équipe s'interroge sur la manière de devenir encore plus efficace**, puis ajuste son comportement en conséquence.	L'environnement d'un projet n'est pas constant; l'équipe agile, qui en a conscience, s'interroge en permanence sur la façon d'améliorer son fonctionnement afin de s'adapter aux nouvelles conditions. C'est aussi l'acceptation du changement.

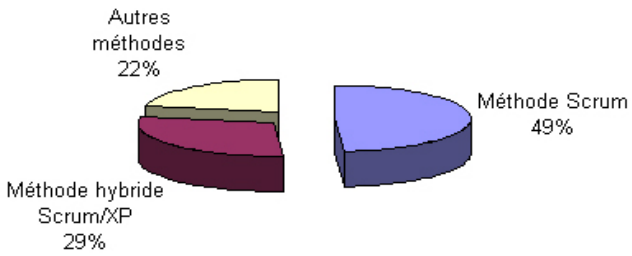
Principales méthodes agiles

En 2009, il existe une panoplie de méthodes agiles, qui sont par ordre alphabétique :

- ASD (Adaptive Sofware Development);
- Crystal;
- DSDM (Dynamic Sofware Development Method);
- RAD (Rapid Application Development);
- Scrum;
- UP (Unified Process);
- XP (eXtreme Programming).

Toutes ces méthodes se nourrissent des **valeurs et principes du Manifeste**; cependant, si elles ont un tronc commun de pratiques, elles se différencient par leur degré de formalisme – le poids de la méthodologie dans la documentation produite, les étapes formelles, les revues de projet, le rythme ou le nombre et la longueur des itérations.

Ce document s'attarde à ne présenter que l'une d'entre elles, la méthode **Scrum**, qui à l'heure actuelle est la plus utilisée et la plus populaire. **Scrum** est en effet une méthode de projet exhaustive, mais simple et aisée à appréhender.



Utilisation des méthodes agiles
Source : CRIM

Gestion agile de projets avec Scrum

Scrum est un terme anglais pour la **mêlée de Rugby**. En effet, cette méthode agile met l'accent sur **l'esprit d'équipe** et sur le fait que tous les **acteurs doivent avancer dans la même direction**. Ce processus s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but, comme c'est le cas en rugby pour avancer avec le ballon pendant une mêlée.

Contrairement à d'autres méthodes agiles comme XP, Scrum ne préconise pas de pratique d'ingénierie. **Scrum** synthétise un flux de travail (« W*orkflow* ») en définissant les **rôles et responsabilités** des intervenants, des **artefacts de communication** et des **réunions**. Cet ensemble permet de rythmer l'avancement du projet, et d'assurer un minimum de communication entre les intervenants.

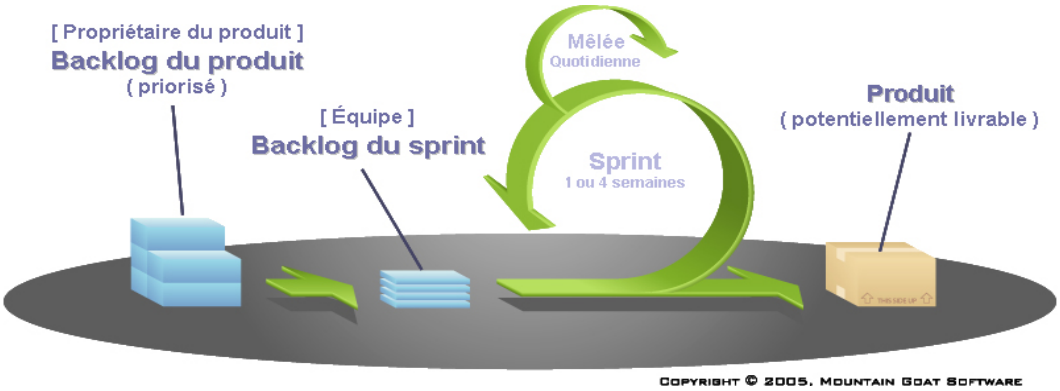
Le principe de base de **Scrum** est de focaliser l'équipe de **façon itérative** sur un ensemble de fonctionnalités à réaliser, dans des itérations de durée fixe de une à quatre semaines, appelées **sprints**. Chaque **sprint** possède un but à atteindre, défini par le « *Product Owner* » (Propriétaire du produit), à partir duquel sont choisies les fonctionnalités à implémenter dans ce **sprint**. Un **sprint aboutit toujours** sur la **livraison d'un produit partiel fonctionnel**.

Une itération contient juste une partie de l'application (web ou autre), mais qui doit être complète : développée et testée.

Un principe fort en **Scrum** est la **participation active du client** pour **définir les priorités dans les fonctionnalités de l'application informatique**, et pour choisir celles qui seront réalisées dans chaque **sprint**. Il peut à tout moment compléter ou modifier la liste des fonctionnalités à réaliser, mais jamais celles qui sont en cours de réalisation pendant un sprint.

Ainsi, les éléments clés de **Scrum** sont les suivants :

- Le client au cœur du projet;
- Esprit d'équipe;
- La communication entre les membres de l'équipe;
- Simplicité, Efficacité, et Qualité ;
- Flexibilité aux changements;
- Avancement basé sur le concret.

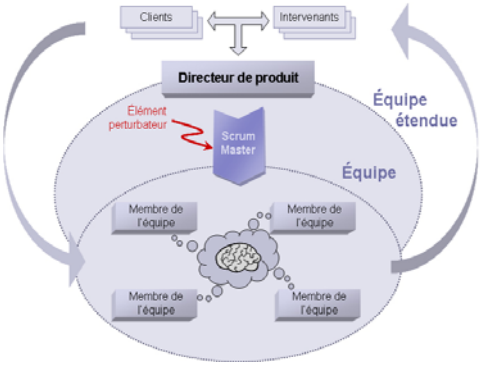


Le vocabulaire Scrum

Scrum est une méthodologie de gestion de projet qui s'articule autour de **trois rôles** (Product Owner, ScrumMaster et équipe), **trois artéfacts**[4] (Product Backlog, User Stories et Backlog de Sprint) et **trois activités** (Sprint planning, Daily Scrum et Revue de Sprint). Cette note de cours s'attarde pricipalement sur les **rôles** et les **artéfacts**. La description des activités sera abordée au prochain cours.

Les trois rôles

Scrum définit trois rôles principaux : le **Propriétaire du produit** (« *Product Owner* »), le **ScrumMaster**, et l'**Équipe**.



- Le **Propriétaire du produit** (« *Product Owner* ») est le représentant des clients et des utilisateurs. C'est lui qui **définit l'ordre et la priorité dans lequel les fonctionnalités seront développées**, et qui prend les décisions importantes concernant l'orientation du projet. Le terme *Propriétaire* n'est d'ailleurs pas à prendre au sens hiérarchique du terme, mais dans le sens de l'**orientation**. Idéalement le **Propriétaire du produit** est un représentant réel du client, toutefois dans un contexte contractuel, ce n'est pas toujours possible. Le cas échéant, ce rôle sera attribué à un membre de l'équipe de projet qui sera un intermédiaire de communication avec le client réel;
    - Le **ScrumMaster** (ou Chef de mêlée) joue un rôle capital: c'est lui qui est chargé de protéger l'équipe de tous les éléments perturbateurs extérieurs. Il s'assure que

[4] Le mot artéfact est employé pour désigner de manière générale un objet ou un produit résultant d'une intervention humaine (par opposition à un phénomène naturel).

l'**Équipe** est complètement fonctionnelle et productive et résout les problèmes non techniques (administratifs ou organisationnels par exemple). Il doit aussi veiller à ce que les valeurs de Scrum soient appliquées. Il n'est pas un chef de projet traditionnel, c'est un *chef de projet 2.0*… avec une gestion de projet collaborative et plus humaine;

- Enfin, l'**Équipe** est généralement constituée de **5 à 10 personnes** et regroupe tous les rôles (développeurs et graphistes). Notons toutefois que les développeurs sont aussi les testeurs MAIS ils doivent « *prouver* » leurs tests (unitaires et d'intégration). De préférence, l'équipe est affectée à plein temps sur le projet et elle est **auto-gérée**. Il n'y a pas non plus de notion de hiérarchie interne : toutes les décisions sont prises ensemble, et personne ne donne d'ordre à l'équipe sur sa façon de procéder. Contrairement à ce que l'on pourrait croire, les équipes auto-gérées sont celles qui sont les plus efficaces et qui produisent le meilleur niveau de qualité de façon spontanée. L'équipe s'adresse directement au **Propriétaire du produit**. Il est conseillé qu'elle lui montre le plus souvent possible l'application développée pour qu'il puisse ajuster les détails d'ergonomie et d'interface par exemple.

Les trois artéfacts

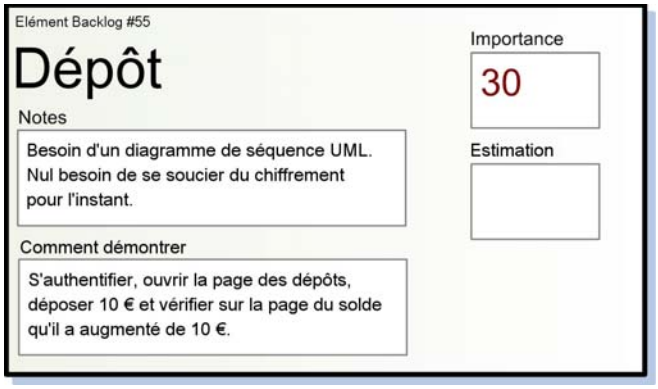
Scrum définit **trois artéfacts** principaux : le « **Product backlog** », le « **Backlog de sprint** » et les « **User Stories** ».

- Le « **Product backlog** », que l'on appelle aussi le « **backlog du produit** », est la **liste** des exigences/ fonctionnalités/livrables à réaliser (« *wish list* ») généralement exprimées par des « **User Stories** ». Le terme « **backlog** » peut être traduit par cahier, liste ou carnet de commande du produit, qui ne collent pas bien avec l'esprit du terme anglais qui évoque aussi une réserve, un retard accumulé, un restant à faire; aussi le terme « **backlog** » a été gardé tel quel. À chaque item du « **backlog du produit** » sont associés **deux attributs** : une **estimation** en points arbitraires, et une **importance**, qui définie la « valeur client » de chaque item. Le but étant de prioriser les différentes fonctionnalités ou item du « **Product backlog** » afin d'implémenter en premier ce qui a le plus de valeur pour le client. Enfin, on y décrit aussi comment la réalisation de chaque item sera démontrée au client avec **un test d'acceptation** ou **« démo »**.

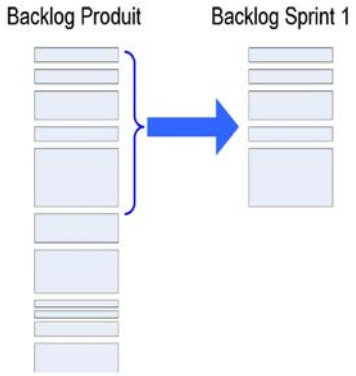
BACKLOG DE PRODUIT (exemple)					
ID	**Nom**	**Imp**	**Est**	**Démo.**	**Notes**
1	Dépôt	30	5	Authentification, ouvrir la page de dépôt, déposer 10€, aller sur la page du solde et vérifier que ça a bien augmenté de 10€.	Nécessite un diagramme de séquences UML. Ne pas se soucier du cryptage pour l'instant.
2	Voir l'historique de ses transactions	10	8	Authentification, cliquez sur « transactions ». Faire un dépôt. Revenir aux transactions, vérifier que le nouveau dépôt apparaît.	Utiliser la pagination pour éviter des requêtes volumineuses . Conception semblable à la page des utilisateurs.

Exemple partiel d'un « backlog de produit ».

Chaque item du « **backlog du produit »** est généralement consignée sur une fiche ou carte. Au prochain cours nous en verrons l'utilité ainsi que la procédure permettant de prioriser les items du « **backlog du produit** ».



- Le « **backlog du sprint** » est un sous-ensemble du « *backlog du produit* » et représente les fonctionnalités affectées à une itération ou Sprint, ainsi que la liste des tâches pour les réaliser. Nous examinerons également cet aspect au prochain cours.



- Les « **User Stories**» (*histoire utilisateur*!) sont des descriptions des items du « **backlog** » formulées en un **maximum de 2 phrases** dans le **langage de l'utilisateur ou du client**. Leur niveau de précision doit permettre à l'équipe de réalisation de les estimer et de les réaliser entièrement au cours d'un *Sprint*. L'avantage des « **User Stories** » est qu'elles facilitent la communication et les échanges avec le client. La rédaction des « **User stories** » prend la forme suivante :

  Comme <type d'utilisateur>, je désire <quoi>, afin de <résultat,valeur ajoutée>.
  (le « quoi » traduisant une fonctionnalité, une tâche ou une action)

  - Ex. 1: pour une application de commerce électronique : « *Comme acheteur, je désire enregistrer mon panier afin de continuer mon magasinage* ».
  - Ex. 2: pour une application Web de recherche d'emploi : « *Comme recruteur, je désire être notifié aussitôt qu'un utilisateur enregistre un profil c'est pourquoi je souhaite en être avisé par e-mail* ».

  - Les « **User Stories** » doivent être **SUFFISAMMENT PETITES** pour être livrées dans un Sprint. Pour découper un projet Web en « User Stories » on peut procéder :
    - **Par étapes d'un Workflow** : L'utilisateur accomplit une tâche selon un workflow bien établi. On découpe les stories par étapes qui seront développées de façon incrémentale;
    - **Par scenario** : On obtient une « User Story » pour le scénario principal, le cas où tout se passe bien, on en a d'autres pour les cas d'erreurs ou les scénarios alternatifs :quand il se passe x ; quand il se passe y;
    - **Par opérations** : Souvent le mode de décomposition le plus évident … Le CRUD (Create, Retrieve, Update, Delete) est un bon exemple ; il est souvent utile de découper ou d'en faire deux en même temps…créer un compte, le consulter, le modifier et le supprimer;
    - **Par type d'entrée, sortie ou configuration** : Des variations d'un point de vue matériel ou non, selon les configurations mais aussi en termes de moyens de saisie. Cela peut se jouer aussi au niveau de l'interface…
    - **Par persona ou rôle** : Cette fois-ci, on décompose les user stories en fonction du rôle et de celui qui va utiliser le produit, le fameux « en tant que… »;
    - **Par niveau de complexité** : Une « User story » va par exemple décrire une fonctionnalité dans son mode de réalisation le plus simple, d'autres suivront par un niveau de complexité plus grand;
    - **Par des critères de performance, de sécurité ou d'utilisabilité** : Ces exigences non fonctionnelles constituent le plus souvent des conditions de satisfaction pour des « User Stories » spécifiques (ex : afficher en moins de 60 sec, cryptées les données, etc.).

- Pour être efficace, les « **User Stories** » doivent être **INVEST...** INVEST étant un acronyme SCRUM pour mémoriser les critères que doit respecter les « User Storries », ainsi :
  - une « User Story » doit être **I**ndependante des autres, , c'est-à-dire qu'elle peut être implémentée indépendamment des autres « User Stories, » déjà implémentées ou non, et qu'elle est aussi indépendante fonctionnellement. Théoriquement, les « User Stories » sont donc interchangeables dans le temps : le critère qui détermine leur ordre d'implémentation n'est pas fonction de pré-requis ou d'interdépendances, mais uniquement des priorités qui leur ont été attribuées;
  - une « User Story » doit être **N**egotiable, c'est-à-dire qu'une « User Story » d'un backlog de produit est modifiable jusqu'à ce qu'elle intègre un backlog de sprint;
  - une « User Story » doit avoir de la **V**aleur pour le client. Cette valeur ajoutée est d'ailleurs l'un des critères qui peuvent être utilisés pour prioriser les « User Stories » (ce que nous verrons au prochain cours...);
  - une « User Story » doit être **E**stimable, c'est-à-dire qu'elle doit être assez claire pour être en mesure d'évaluer la charge de travail nécessaire à sa réalisation;
  - une « User Story » doit être **S**mall, qu'elle elle doit être réalisable en totalité dans un sprint de 2 à 3 semaines.Plus une « User Story » sera grande, plus il y aura d'incertitude pour l'estimer....
  - enfin, une « User Story » doit être **T**estable, c'est-à-dire qu'on doit pouvoir écrire des tests d'acceptation pour valider son implémentation. Envisager et prévoir les tests dès les spécifications permet de lever très tôt la plupart des ambiguïtés....

Médiagraphie

AGILE QUÉBEC. *Bienvenue sur Agile Québec*, [En ligne] http://www.agilequebec.ca/ (Page consultée le 27 février 2009).

CROSSBOW-LAB. *Exteme Programming (XP)*, [En ligne] http://www.crossbowlabs.com/dossiers/extreme-programming, (Page consultée le 27 février 2009).

KNIBERG, Henrik. *Scrum et XP depuis les Tranchées - Comment nous appliquons Scrum*. C4Media Inc, 2007, 160 pages, ISBN: 978-1-4303-2264-1

MESSAGER ROTA, Véronique. *Gestion de projet – Vers les méthodes agiles*, Paris, Édition Eyrolles, 2008, 247 pages, ISBN : 978-2-212-12165

WIKIPEDIA. *Scrum (Méthode)*, [En ligne] http://fr.wikipedia.org/wiki/Scrum_%28m%C3%A9thode%29 (Page consultée le 19 octobre 2012).