# Chapitre 5
# Construire et prioriser le Product Backlog

## 1. Pourquoi investir dans le Product Backlog ?

Que l'on soit en contexte Agile ou pas, il est assez évident que pour réussir un projet nous devons posséder une bonne vision du système ou produit que l'on va développer, par le biais d'une définition pertinente du besoin auquel il répond.

En Scrum donc, pas de projet réussi sans un Product Backlog bien construit et intelligemment priorisé. Pour ce faire, évidemment, le Product Owner joue un rôle fondamental, par sa connaissance du métier et/ou du marché, mais aussi par sa capacité à transcrire efficacement, découper et ordonner ce qui est attendu par les utilisateurs du logiciel.

Là où nous avions l'habitude en tant que « Client » d'écrire les spécifications fonctionnelles détaillées faisant office d'expression des besoins, nous allons devoir nous familiariser avec une nouvelle méthode. Cette familiarisation demande un temps d'apprentissage et se doit d'être partagée par l'ensemble des parties prenantes du projet, ce qui signifiera formation, assimilation et mise en pratique des nouvelles techniques.

Dans ce chapitre, nous allons donc parcourir les concepts et méthodes permettant de construire, ordonner, affiner et gérer au quotidien le Product Backlog.

## 2. La brique de base du Product Backlog : la User Story

Aussi étonnant que cela puisse paraître Scrum ne prescrit pas un format ou un contenu précis du Backlog ! Pour autant, un formalisme s'est imposé et on le retrouve quasiment systématiquement sur tous les projets Scrum : il s'agit de la notion de **User Story**, qui est empruntée à XP.

Une User Story est une description simple et compréhensible d'un élément de fonctionnalité à valeur « métier » du système. Elle est donc bien exprimée du point de vue de l'utilisateur. Et elle est guidée par la réponse à ces trois questions :

– Qui fait la demande ou qui bénéfice de la demande ? (rôle utilisateur)

– Quelle est la demande ? (le besoin)

– Quelle valeur métier découle de la réalisation de ce besoin ?

Ci-dessous quelques exemples (nous verrons plus loin comment bien rédiger les User Stories) :

« Je souhaite que la TVA soit automatiquement calculée sur les factures ».

« Je souhaite pouvoir supprimer les clients n'ayant pas passé de commandes depuis plus d'un an ».

En complément, on emploie aussi la notion d'**Epic Story**. On peut considérer une Epic comme une « **macro User Story** », c'est-à-dire qu'elle englobe dans sa définition un sous-ensemble de User Stories.

Par exemple, en relation avec les User Stories décrites précédemment, nous pourrions avoir les Epics suivantes :

« Je souhaite que mes factures soient établies automatiquement ».

« Je souhaite avoir une gestion de mes clients ».

C'est donc l'ensemble des Epic et User Stories que constitue le Product Backlog.

Il est en pratique utile de regrouper les Epic ou User Stories (en particulier pour la priorisation) : pour ce faire, on emploie communément les concepts de **Thèmes** ou **Activités**, qui sont des regroupements thématiques de Stories.

# 3. Comment rédiger les User Stories et Epics ?

## 3.1 Règle des 3C

Pour guider la rédaction des User Stories, un principe très simple à retenir a été proposé par Ron Jeffries. Il s'agit de la règle des 3C :

| | |
|---|---|
| **Carte** | La Story est écrite sur une carte de taille assez réduite. Ces fiches peuvent être annotées (estimation, etc.). |
| **Conversation** | Les détails de la Story seront exprimés lors de conversations avec le Product Owner. |
| **Confirmation** | Des tests de validation sont décrits avec la Story (ils serviront à valider qu'elle a été réalisée correctement). |

En pratique, on écrira donc la Story sur une petite carte de couleur colorée épinglée ou collée sur un tableau (principe hérité du Kanban, si vous vous souvenez bien), sous la forme suivante :

– On commence avec un titre.

– On ajoute une description synthétique de la « tâche utilisateur » et de ses objectifs.

– On peut y ajouter toutes les notes, dessins ou informations utiles. Exemple : chiffrage, indicateur de priorité ou valeur métier.

– On y ajoute idéalement les critères de validation (par exemple au dos si on n'a plus de place…).

Cela donnera, dans la forme la plus simple, quelque chose qui ressemble à ceci :

La carte est un concept de partage de l'information extrêmement synthétique et efficace, mais dès que des données additionnelles s'ajoutent, au fur et à mesure de l'analyse, attention à ne pas surcharger celle-ci. On voit assez vite venir la nécessité de passer par un outillage plus sophistiqué (et informatisé) pour gérer l'information : nous parlerons de cela plus loin…

## 3.2 Rédiger une bonne User Story : le principe INVEST

Lorsqu'on entre dans le processus de rédaction, on peut rapidement être perdu… Dans ce cas, un autre principe très utile vous guidera : il s'agit du principe INVEST.

Il indique qu'une bonne User Story doit être :

- **Indépendante** des autres histoires d'utilisateur (dans la mesure du possible).
- **Négociable** : elle doit pouvoir être discutée avec l'équipe chargée de la réalisation du produit, notamment lors de l'estimation.
- Source de **Valeur** : elle doit être porteuse d'une valeur pour le client ou l'utilisateur.
  - Une User Story ne décrit pas des finalités techniques !
- **Estimable** : elle peut être estimée par l'équipe de réalisation avec un risque d'erreur faible.
  - Elle doit, à cette fin, être rédigée de manière claire et compréhensible.
- D'une taille **Suffisamment petite** afin de faciliter son estimation et afin d'assurer qu'elle puisse être conçue, développée et testée au sein d'un Sprint.
  - Si la Story est trop grosse, c'est sans doute plutôt une Epic, qui devra être découpée plus finement préalablement à la réalisation.
- **Testable** : une User Story doit être accompagnée des critères de validation permettant sa validation.
  - Nous verrons dans le chapitre dédié aux tests comment formaliser cela.

## 3.3 Erreurs courantes

Pour bien rédiger nos Stories, il est aussi très instructif de savoir quelles erreurs éviter :

– Trop détailler la description et rentrer trop tôt dans un niveau de détail fin.

  N'oublions pas qu'on ne cherche pas à faire les spécifications détaillées complètes préalablement au développement comme dans les méthodes traditionnelles !

– Perdre la notion utilisateur dans la description ou utiliser des acteurs trop génériques : cela peut créer des ambiguïtés et imprécisions

  *Exemple* :

  – En tant que client abonné, je veux pouvoir consulter les tarifs des billets d'avion.
  – En tant que client standard, je veux pouvoir consulter les tarifs des billets d'avion.

  *Plutôt que* :

  – En tant qu'utilisateur, je veux pouvoir consulter les tarifs des billets d'avion.

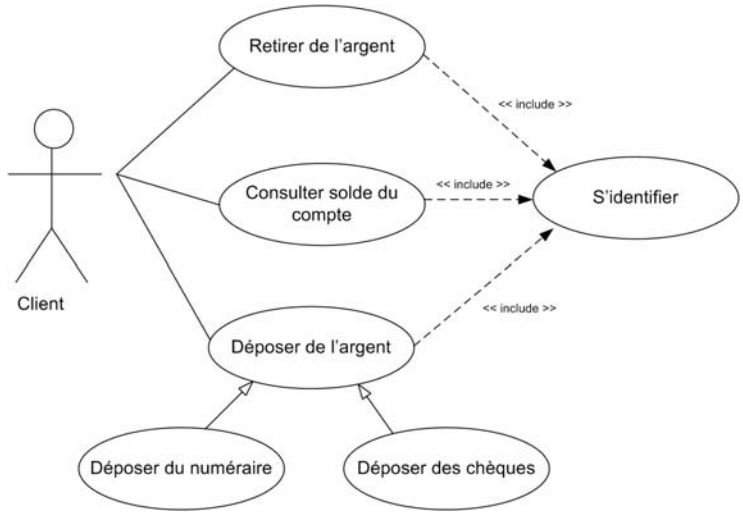
  En effet, on voudra vraisemblablement proposer des offres ou des tarifs différents aux abonnés par rapport aux autres personnes qui consultent les tarifs, ce qui n'apparaît pas dans la seconde description.

– Partir d'un cahier des charges rédigé en amont et le découper en User Stories en s'appuyant aveuglément sur sa structure textuelle. Cela peut être une approche commode pour une équipe débutante mais n'est pas une pratique recommandée (nous verrons plus loin une méthode pratique pour initialiser le Product Backlog).

– Vouloir avoir un niveau de détail constant. Ce point est très important, car il est au cœur de la démarche Agile. Le contenu des User Stories est amené à évoluer au fil du temps, au fur et à mesure de l'affinage de la compréhension du besoin et de l'analyse. Typiquement, une Story dont la réalisation est prévue dans plusieurs Sprints ne sera décrite que de manière très simple (titre, description synthétique), alors qu'une User Story dont la réalisation est proche doit être complétée par des tests de recette, des exemples, des règles de gestion, des maquettes d'écran, etc.

– Assimiler une User Story à un **Use Case**. Bien que la comparaison entre les deux notions soit souvent utile, ce sont deux approches aux caractéristiques sensiblement différentes.

Sans rentrer dans un niveau de détail trop fin, on peut dire qu'une modélisation à base de Use cases est une description de processus (par définition, un Use case représente une séquence d'actions qu'un système ou toute autre entité peut accomplir en interagissant avec les acteurs du système), qui s'appuie souvent sur des diagrammes UML, comme dans l'exemple ci-dessous :




## 3.4 La Story technique : solution ou aveu d'échec ?

Disons en préambule que ce thème est éminemment polémique dans la communauté Agile, car certains considèrent que le Product Backlog ne doit contenir que des éléments qui ont de la valeur d'un point de vue utilisateur ou métier alors que d'autres disent que finalement Scrum ne dit rien de précis à ce sujet, donc on peut inclure des sujets à vocation purement technique dans le Backlog.

# Chapitre 1-2
# L'esprit agile

## 1. Introduction

Maintenant que nous avons fait le tour du cadre Scrum, il nous reste à voir le plus difficile : acquérir l'esprit agile !

Si on vous explique les règles de jeu pour jouer au football, vous comprendrez sans problème mais cela ne signifiera pas que vous serez un excellent joueur. Il vous faudra sans doute des années de pratique pour devenir très bon. Avec Scrum, c'est pire ! Ses règles sont très simples à comprendre mais vous pouvez passer des années à le pratiquer sans pour autant acquérir l'art de conduire des projets agiles. Pourquoi ? Parce que pour exceller dans Scrum ou toute méthode agile, l'excellence ne se gagne pas durant les années de pratique, mais dans l'état d'esprit dans lequel on veut pratiquer !

Qui dit esprit, dit personnalité. Nous savons combien il est difficile pour chacun de nous de se remettre en question et de changer ses habitudes. L'agilité va mettre à rude épreuve les habitudes de travail. Si vous négligez ses valeurs, ses règles de conduite, sa façon de penser, vous n'avez pas l'esprit agile pour intégrer une équipe agile.

Voici une définition de ce qu'est l'agilité :

***« L'agilité est la capacité d'une organisation à fournir, tôt et fréquemment, des services de qualité impactant les utilisateurs, tout en s'adaptant à temps aux changements qui peuvent survenir dans l'environnement. »***

Quelque chose nous déplaît dans cette définition. Elle ne fait pas ressortir le poids de l'individu. L'agilité appliquée sans prendre en compte les valeurs humaines qu'elle tente d'améliorer n'est plus de l'agilité mais un process managérial supplémentaire. L'agilité invite à une remise en question radicale de notre façon de travailler ensemble, que ce soit avec les outils ou dans la relation humaine. Le cœur d'un projet agile ne doit pas se focaliser uniquement sur les méthodes ou les outils mais aussi et surtout sur les individus...

Pour y parvenir, l'agile vous invitera à appliquer systématiquement de façon empirique :

– **La transparence (visibilité de l'avancement à tous les niveaux)**
– **L'inspection (deux jalons suffisent dans Scrum)**
– **L'adaptation (ajustement dès que possible pour minimiser le risque)**

L'empirisme affirme que la connaissance provient de l'expérience. Au lieu de développer une application durant 6 mois pour ensuite présenter le résultat au client, nous allons plutôt avancer de façon transparente, petit à petit, sprint par sprint, release par release. Durant le sprint, chaque jour, l'équipe technique fera un bilan rapide de la situation, et en fin de sprint une démonstration sera faite au métier, du travail effectué durant le sprint. Ces deux jalons (état des lieux quotidien et démonstration en fin de sprint) suffisent largement pour s'assurer que nous avançons dans la bonne direction et garantissent qu'il est possible d'intervenir rapidement en cas de déraillement.

Nous appliquons systématiquement l'inspection et l'adaptation dans le cycle des sprints.

La transparence permet à tous les intervenants du projet d'avoir une parfaite visibilité sur l'avancement du projet. Ici encore, il ne faut pas s'encombrer avec des outils compliqués, un simple tableau de bord affiché au mur suffira. Des discussions face à face permettront de gagner la confiance des uns et des autres. Depuis une décennie, les outils de management visuel évoluent très vite pour offrir plus de créativité, de flexibilité, de fluidité, de rapidité, d'agilité ! Mais au-delà des outils, il y a les individus. Tout le management visuel a pour unique objectif de diffuser un ensemble de méthodes et de règles de conduite pour réorganiser la communication entre les individus. Pour que cette révolution fonctionne, tout ce qui est mis en place doit être simple, clair et absolument transparent.

Dans mes différentes missions, j'ai toujours rencontré ce souci avec ceux qui débutaient en agilité : ils préféraient continuer à utiliser des outils inappropriés à culture agile et refusaient les Post-it, prétextant qu'on ne peut manager de gros projet juste avec des Post-it ! Pourtant, des géants du CAC 40 par exemple nous ont prouvé depuis longtemps que c'était possible. Il ne sert à rien de brusquer les choses, il faut du temps et de la patience pour changer les mentalités, cela peut prendre trois mois, six mois ou même des années selon la résistance au changement.

Dans tous les cas, l'agilité n'est pas réservée qu'au développement, il peut s'appliquer au niveau managérial pour conduire ses projets avec les mêmes activités qu'autrefois, mais de façon différente. Même la RH peut se mettre à l'agilité ! Du moment que vous savez identifier des actions dans vos processus métier, elles peuvent prendre la forme d'US. Donc, quelle que soit la taille des projets, tout peut être décomposé en actions. Qu'est-ce qui différencie un gros projet d'un petit projet en agilité ? Le nombre de Post-it utilisés !

Le plus difficile dans les méthodes agiles est donc le changement des mentalités. Pour nous y aider, les fondateurs de ce mouvement révolutionnaire ont créé un Manifeste regroupant douze règles d'or à appliquer absolument pour se prétendre agile ! Manquer à une de ces règles ne vous donnera pas l'esprit agile. Refuser d'appliquer un de ces douze commandements, c'est risquer d'être expulsé dans « l'enfer » des cycles V, l'horreur absolue pour tout agiliste qui se respecte.

**En fédérant les méthodes agiles (comme XP, Kanban, Scrum…), le Manifeste agile constitue l'acte de naissance du mouvement de l'agilité qui a pris de l'ampleur depuis quelques années.**

Regardons de plus près ces douze principes sacrés, imprimés éternellement à l'origine, sur deux pages, par les dix-sept apôtres de l'agilité.

## 2. Le Manifeste

### 1. Prioriser la satisfaction du client

C'est le principe le plus important. Tout est fait pour y parvenir, notamment en livrant au client rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.

### 2. Accepter les changements

Cela ne veut pas dire que l'on doit tout bouleverser à la moindre demande. Par contre, on accepte que des changements soient possibles et c'est à nous de rectifier les choses sans stresser.

### 3. Livrer en permanence des versions opérationnelles du produit

Chaque sprint doit être couronné d'une ou plusieurs livraisons de fonctionnalités.

### 4. Assurer au maximum une coopération entre l'équipe du projet et les gens du métier

Parlez-vous ! Parlez-vous ! Échangez, clarifiez, demandez… Discutez, discutez et discutez toujours sans aucune frontière.

### 5. Construire les projets autour de personnes motivées

Transférer les membres d'une équipe à l'autre brutalement peut casser la motivation. L'esprit d'équipe est primordial ! Fournissez à l'équipe tout le soutien dont elle a besoin, faites-lui confiance pour atteindre les objectifs fixés. Par-dessus tout, bannissez l'esprit de chef, l'équipe sera au top si elle s'autogère et baigne dans les joies de l'autonomie.

### 6. Favoriser le dialogue direct

Privilégiez toujours la communication en présentiel et le face à face ! Vos US, c'est du face à face pour les comprendre. Vos problèmes, c'est du face à face pour les résoudre. Vos reportings, c'est encore du face à face, et de même vos succès, c'est toujours du face à face pour les fêter. Discutez en toute occasion, fini les bureaux isolés où l'on pouvait passer des jours sans discuter avec les membres du service.

### 7. Mesurer systématiquement l'avancement du projet en observant l'application produite

C'est bien de produire de nombreuses fonctionnalités mais cela ne suffit pas si celles-ci reviennent sous la forme de bugs. Un logiciel parfaitement opérationnel sera toujours la meilleure mesure d'avancement.

### 8. Adopter un rythme constant et soutenu

Si l'équipe travaille toute la nuit et se félicite ensuite d'avoir été réactive, ce n'est pas agile. Idem si on produit des tas de fonctionnalités et que le PO n'a pas le temps de valider. L'agilité encourage un rythme de développement soutenable. Les commanditaires, les développeurs et utilisateurs devraient tous ensemble maintenir un bon rythme constant pour le bien de tous.

### 9. Contrôler continuellement l'excellence de la conception et la bonne qualité technique

Il faut faire de la conception avant de coder. La qualité n'est pas négociable en agilité ! Une bonne architecture est toujours ouverte aux extensions mais fermée à des modifications rocambolesques pouvant mettre en péril le projet.

### 10. Privilégier la simplicité en évitant le travail inutile

Ce n'est pas évident de rester simple, surtout si on veut se sentir indispensable et plus intelligent que les autres... C'est quoi exactement savoir rester simple ? C'est l'art de minimiser la charge de travail inutile tout en restant efficace.

### 11. Auto-s'organiser et responsabiliser les équipes

Une équipe travaillant sous la contrainte sera toujours moins performante, le codage informatique nécessite énormément de concentration. Laissez les équipes gérer leurs domaines de compétence. Les meilleures architectures, spécifications et développement émergent d'équipes auto-organisées et responsables.

### 12. Améliorer régulièrement l'efficacité de l'équipe en ajustant son comportement

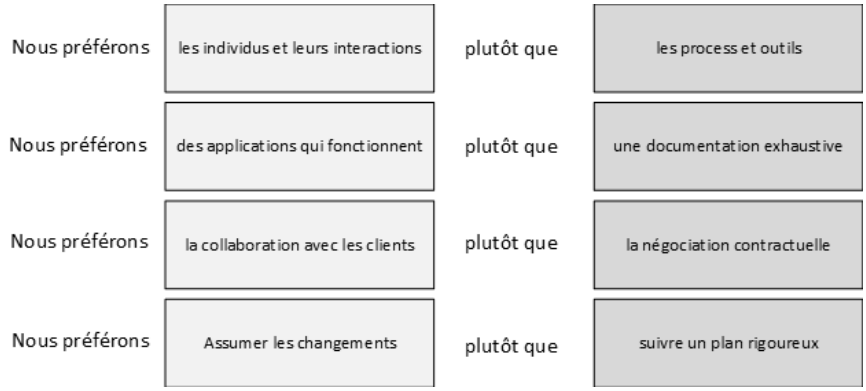
Ne surtout pas négliger la Retrospective ! C'est à ce moment que l'équipe se remet en question pour toujours progresser. Tout le monde doit prendre la parole et personne n'a à influencer qui que ce soit. Pour devenir plus efficace, on a besoin de l'avis de tout le monde.

Dans ce Manifeste, il apparaît que la communication est de rigueur et, comment voulez-vous communiquer convenablement avec les autres si vous adoptez une posture de chef ou bien sûr, si vous manquez de respect à vos collègues ? Nous le savons, un vrai chef n'est pas celui qui s'impose comme tel mais celui qui est reconnu comme tel, celui qui a prouvé qu'il disposait de certaines qualités et valeurs appréciées par l'équipe… Il n'y a rien de pire qu'un chef qui joue les esprits ouverts alors qu'il est, au fond, ancré sur ses convictions. En général, ce genre de chef ne supporte pas qu'on lui tienne tête, il ose l'impensable car il se sait protégé par son statut. Cette attitude n'est pas l'apanage des chefs : vos propres collègues peuvent tout autant être nocifs pour l'équipe. La bonne pratique de l'agilité exige certaines qualités humaines.

Outre le Manifeste, nous disposons de quatre valeurs indispensables pour comprendre les priorités accordées dans l'agilité.

## 3. Les quatre valeurs

Dans la pratique de l'agilité, on reconnaît les valeurs ici à droite, mais on privilégie avant tout les valeurs affichées à gauche :




### 3.1 Préférence sur les individus et leurs interactions

Il faut tout faire pour simplifier les process et outils afin de favoriser la communication directe entre individus. Les process sont souvent lourds et cette lourdeur administrative est incompatible avec l'esprit d'adaptation au changement qu'exige l'agilité.

Côté outil, il en est de même. Il n'est pas acceptable de ne pas pouvoir avancer correctement à cause d'un outil trop complexe, trop instable : passer une heure à déboguer l'outil de versioning de code alors que cette heure aurait pu être consommée dans le codage, ce n'est pas rentable ! Que l'objectif du sprint ne soit pas atteint car 30 % du temps ont été gaspillés à rendre stable tel outil n'est pas non plus acceptable ! Il faut discuter, discuter et discuter encore, c'est cette communication face à face qui nous fera vraiment avancer ; si un outil dérange vraiment, il faut le changer. Aujourd'hui, nous avons l'embarras du choix, de nombreux outils sont disponibles sur le marché du logiciel. Un outil comme **Klaxoon** suffit largement pour construire une roadmap, faire du Story Mapping, alimenter un backlog, les plans de release ou les sprints. Plus l'outil est simple et visible par tous, plus les membres de l'équipe pourront passer de temps entre eux à échanger sur l'essentiel et plus vite l'équipe sera efficace.