

SCRUM

**Le guide pratique
de la méthode agile la plus populaire**



Plus de
15 000
exemplaires
vendus

Claude Aubry

Préface de Pablo Pernot

4^e édition

DUNOD

© Dunod, 2015

ISBN : 9782100742165

5, rue Laromiguière, 75005 Paris

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Dessins de Patrice Courtiade

Illustration de couverture :
© iStock.com / PeopleImages

Visitez notre site Web : www.dunod.com/

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher, Dunod, 5 rue Laromiguière, 75005 Paris www.dunod.com.

Préface

Scrum est un drôle de truc. Une petite chose devenue énorme. Quelques paragraphes qui révolutionnent nos méthodes de travail, voire nos vies. Quelques paragraphes... Initialement, *Scrum* est une méthode pour réaliser des projets dans des environnements complexes, incertains. Cette petite idée a fait son chemin : elle a détrôné l'aristocratie méthodologique en place (ouste les PMI, CMMi, RUP, etc.). Les géants aux pieds d'argile se sont effondrés. Comme si on avait ouvert une fenêtre dans une pièce fermée depuis bien trop longtemps. Aidé en cela par l'époque, cette époque moderne et incertaine à la concurrence exacerbée où toutes les énergies comptent. J'en connais qui se marrent bien : ils ont soulevé avec Scrum le voile sur tous les petits mensonges de nos organisations.

C'est drôle aussi, le Petit Poucet est devenu l'ogre. Scrum est comme un trou noir qui avale tout ce qui passe à côté de lui. Le livre de Claude ne parle pas que de Scrum, mais de tout ce que Scrum symbolise et a embarqué avec lui dans sa proposition d'une organisation ou d'un projet agile. Et c'est beaucoup plus que les quelques paragraphes initiaux. Sans complexe Scrum a emprunté là où était la valeur. Dans notre bon sens, dans les autres méthodes agiles, et autour encore quand cela en valait la peine. *Extreme programming*, c'est comme l'oncle rustre et frustré. Il ronchonne dans son coin, Scrum ne serait qu'un vendu qui aurait oublié certaines de ses valeurs. *Extreme programming* est ombrageux et exclusif. Mais il oublie que lui et Scrum sont de la même famille, et que leurs discours sont proches. Scrum est juste un beau parleur, et cela génère des jalousies. De fait, Scrum rayonne aujourd'hui. J'en connais qui se marrent bien quand ils découvrent que Scrum est devenu la méthode de référence.

C'est drôle, car les gens se trompent sur Scrum. Ils le pensent simple, souple et facile. Bien compris et mis en œuvre il se révèle complexe, rigoureux et ardu. C'est tout le paradoxe de nos méthodes : elles sont comme nos organisations ou projets, complexes. Il faut les adapter, se focaliser sur la valeur, s'améliorer, essayer, apprendre par l'échec. Ce paradoxe est partout chez Scrum. Comme évoqué, la petite méthode est devenue grande, David, Goliath. L'approche alternative est devenue la référence. On se jette sur elle car on a bien compris qu'elle était la mieux armée pour lutter dans le monde actuel. Mais sa légèreté est trompeuse, car c'est bien une lame tranchante, un outil aiguisé pour appliquer les principes et valeurs de l'agile. Beaucoup de gens se trompent de sens et l'attrapent par la lame !

Le succès de cette méthode pourrait aussi la rendre obèse et caduque. En cela le livre de Claude est une sorte d'antidote, une base saine. On commence d'ailleurs à payer le retour de flamme de l'agilité : de trop nombreuses incompréhensions, récupérations font la promotion d'un agile complètement dévoyé. C'est exactement ce qui est arrivé au Lean des années 1990. Donc pour faire sérieux, pour ne pas dire « agile » qui fait peur au système en place (mais oui, car il décrète la fin du système précédent), on parle beaucoup de Lean (un autre grand-oncle, mais bien plus vieux). Ce n'est pas une bonne raison. C'est encore une fuite. D'autant plus que le Lean que l'on évoque est archaïque. On évoque le

Lean du non-gaspillage : on oublie complètement celui du respect des personnes et de l'amélioration continue. Le Lean de la manufacture ou de l'industrie n'est pas fait pour nos organisations ni nos projets modernes, ne l'oubliez pas. La période n'est plus au Lean, standardisation, amélioration continue sur l'élimination des gaspillages par implication et respect des personnes. La période est à l'adaptation, à la NON-standardisation, à la NON-linéarité, à la NON-répétabilité, induites par la complexité de notre temps. C'est bien cela aussi ce paradoxe : probablement aucun des lecteurs du livre de Claude n'appliquera la même saveur de Scrum. Et pourtant parle-t-on bien de la même chose ? Oui, mais tout est contexte et intention. Il faudra essayer en puriste avant de changer, mais impossible d'essayer sans s'adapter. Ah ce joli paradoxe ! Une chose est sûre, je me marre quand je vois l'ancien système qui remue, se secoue encore un peu, tente de résister. Mais on ne résiste pas à l'irrésistible progression du temps et du contexte. Vive Scrum (et son oncle ronchon *Extreme Programming*, ou son cousin ambitieux Kanban) !

D'ailleurs Claude est aussi un drôle de gars. Il fallait bien cela pour porter cette synthèse de Scrum (et au-delà) depuis cinq années à travers ce livre. Vous l'avez compris ce livre ne se résume pas au Scrum officiel, mais bien à sa pratique vivante qui absorbe, essaye, rejette, intègre, les bonnes idées, les bonnes pratiques des dix, vingt dernières années. Vous pensez Claude doux, simple et scolaire ? Il est taquin, curieux et rigoureux (mais c'est vrai qu'il est doux et sage), il est aussi complexe que l'approche qu'il prône (et aussi sain). On aborde Claude comme Scrum, avec facilité, c'est une maison accueillante. En creusant on y découvre des choses inattendues. Une soif d'apprentissage (qu'il diffuse : c'est pour cela qu'il est un si bon pédagogue je pense), et une vraie curiosité : il veille. En fait c'est cela, c'est un veilleur. Il veille à la cohérence de cette pensée qui ne cesse de se développer ; pas comme un gardien d'un temps passé, mais comme le porteur d'une nouvelle perspective, car il prospecte sur ces nouvelles approches, ces nouveaux flux d'idées, de façon intarissable (une quatrième édition, ceci explique cela, Claude est un prospecteur, un pionnier). Moi je me marre bien avec Claude, quand je le vois vitupérer contre un discours sans nouvelles idées, ou quand — d'un petit geste de la main très révélateur — il laisse les pissoirs maugréer dans leur coin.

Avant de vous laisser avancer avec Claude, et de vous souhaiter une bonne lecture, je voudrais rappeler un point qui me paraît essentiel d'avoir en tête lors de cette découverte du mouvement Agile. Ce point c'est la zone d'inconfort. Si vous vous lancez dans ces pratiques avec facilité c'est que vous vous plantez probablement. L'agile propose un changement de paradigme assez radical par rapport à notre manière de percevoir l'entreprise, sa création de valeur et sa dynamique de groupe. Nous sommes aux antipodes des habitudes françaises des cinquante dernières années : son cartésianisme, sa hiérarchie, et (contrepartie de sa sophistication que j'aime) son culte de la perfection.

Prenez ce livre comme si il y avait un sticker « Positionnement dangereux » dessus. Et mettez-vous en danger pour bien comprendre les tenants et les aboutissants de l'Agilité. Sans danger, sans inconfort, c'est probablement que vous maquillez vos anciennes habitudes sous couvert de nouveaux habits. Et Claude aura échoué.

Essayez de provoquer le « Oh ! » des familles américaines qui, dans les années cinquante, ont vu sur leurs TV débouler Elvis avec son déhanchement et son magnifique « *That's all right mama* ». Si vous ne ressentez pas le toupet qu'il faut pour penser et être agile aujourd'hui c'est que : a) nous sommes en 2030 et vous tenez entre les mains une très vieille édition du livre de Claude, b) vous faites partie des rares personnes pour qui l'agile est innée c) alerterez-vous et mettez-vous dans l'inconfort, prenez plus de risques.

Quand vous essayez une pratique de ce livre, faites la réellement, pour voir. Pour en connaître les limites et les véritables enseignements. Presque jusqu'à l'absurde, qui pourrait se révéler plus sain qu'il n'y paraît. Soyez « jusqu'au-boutiste » pour savoir, pas dogmatique pour fossiliser. Je ne parle pas de s'enfermer dans quelque chose d'extrême, mais d'essayer vraiment, puis de placer son curseur à bon escient, en connaissance. Beaucoup voient dans Agile du bon sens, c'est en grande partie vrai. Mais en grande partie seulement, un tiers de son approche n'est ni intuitive, ni apparentée au bon sens. Souvent on oublie celle-là, et la cohérence générale en pâtit. Pour découvrir ce paysage secret, il faut s'y balader.

Aujourd'hui les mots « agile », « lean », « lean startup », « design thinking », « entreprise libérée » sont lancés. Chacun de ces mots est un emballage qui correspond le mieux à la population à laquelle il s'adresse (*Agile* pour les informaticiens, *Lean* pour les méthodologues, *Lean Startup* pour les métiers, *Design Thinking* pour les agences et créatifs, entreprise libérée pour les entrepreneurs). Mais derrière c'est le même mouvement de fond : cette transformation profonde, révolutionnaire, sur la façon de percevoir et de penser nos organisations et nos relations. Est-ce nous qui l'espérons tant que nous en faisons une réalité, ou cette transformation est-elle réellement inéluctable (ce que j'espère) ?

Comment allez vous juger que cela marche d'ailleurs ? Scrum porte-t-il ses fruits ? Ne lisez pas votre implémentation en référence aux mesures de l'ancien système. Essayez de savoir ce qui se raconte dans les repas d'amis et de familles le soir concernant votre organisation, votre utilisation de Scrum, le meilleur indicateur se trouve là.

Mince je suis en train de vous dire que vous imaginez Scrum simple, qu'il ne l'est pas, que sans inconfort, prise de risque, point de salut, Vous êtes découragés ? Bien ! Maintenant tout ne peut qu'aller mieux.

Pablo PERNOT
Agent provocateur, auteur du blog Are you agile

PS : Claude et moi avons un autre centre d'intérêt commun : la musique, le rock'n'roll, et s'il fallait se focaliser sur un groupe : Led Zeppelin. Ça doit compter pour chambouler les idées, d'aimer ces quatre gars qui débarquaient comme une horde sauvage sur la scène et assenaient le riff inégalé de « *Whole Lotta Love* ». Pour avoir donc une expérience améliorée de la lecture de ce livre, je vous recommande donc de l'accompagner avec un disque de Led Zeppelin dans les oreilles et une Chartreuse (ou une mirabelle) sur les lèvres.

Avant-propos

Quand j'ai achevé la troisième édition en mai 2013, je ne croyais pas que j'écrirais un jour une quatrième. Je pensais sincèrement qu'il n'y aurait plus rien à dire sur Scrum. Je me trompais.

On pourrait croire que cette nouvelle édition est due à une évolution majeure du « Scrum officiel ». Mais non ! Et pourtant dans la partie du livre qui présente le cœur de Scrum, j'ai tout de même opéré de nombreux changements :

- Le premier chapitre « Scrum dans le mouvement agile » (1) a été complètement réécrit, à la fois parce qu'il y a eu du « mouvement » bien sûr, mais aussi parce que la place de Scrum dans l'agilité s'est, à mes yeux, éclaircie.
- Un nouveau chapitre apparaît, il s'appelle « Les gens de Scrum » (3). Je parle plus des gens dans cette édition et pas seulement du « Product Owner » (4) et du « ScrumMaster » (5).
- Le chapitre sur le backlog s'est « décomposé » en « Structurer le backlog » (6) et « Affiner le backlog » (7). De mon point de vue, l'affinage, pratique encore émergente, est devenu une notion de premier ordre.
- Le chapitre « Définition de fini » (8) a changé de place, il arrive plus tôt, pour lui donner plus d'importance dans le déroulement du sprint. Il est accompagné de sa petite sœur, la définition de prêt, une pratique émergente.
- Tous les autres chapitres de cette première partie qui va jusqu'au chapitre 12 ont été remaniés.

À ce propos, je conseille aux auteurs d'une quatrième édition de ne pas hésiter à réécrire plutôt qu'essayer d'améliorer un texte qui a déjà subi plusieurs passages d'écriture.

Maintenant que Scrum s'est largement diffusé, je m'adresse, dans cet ouvrage, non seulement à des débutants, mais aussi à ceux qui ont déjà pratiqué.

Cette première partie du livre s'adresse à tous. Je conseille de tout lire, dans l'ordre des chapitres. Certaines parties de ces chapitres sur les pratiques avancées pourront faire l'objet d'une seconde lecture au moment où on essayera de les mettre en œuvre sur le terrain.

La deuxième partie du livre commence par le chapitre 13 « Contextualiser Scrum » ; il donne les clés pour la suite, qui porte sur l'écosystème Scrum, tout ce que Scrum attire dans son « cadre ». Cette quatrième édition reflète les évolutions de cet écosystème :

- Les chapitres « Découvrir le produit » (14) qui a été repensé et « Raconter la story » (15), qui est nouveau, permettront au lecteur de connaître la définition de produit « agile ».
- Le chapitre « Planifier la release » (16) était placé plus tôt dans les versions précédentes. Complètement revu dans l'esprit et dans la forme, il a maintenant sa

place dans les compléments de « gestion de projet », avec « Tirer profit des outils » (17) et « Améliorer la visibilité avec les indicateurs » (18).

- Deux chapitres présentent des pratiques issues de deux autres méthodes agiles, XP (19) et Kanban ; « Appliquer Kanban à Scrum » (20) est tout nouveau.
- Les deux derniers chapitres m'ont demandé beaucoup d'efforts. Je voulais rester simple et concis sur des sujets qui pourraient, à eux seuls, faire l'objet d'ouvrages entiers. Ils ne sont pas seulement renommés en « Développer un produit avec plusieurs équipes » (21) et « Transformer les organisations » (22), ils ont été totalement réécrits dans cette édition quatre.

Autres nouveautés

- Les références bibliographiques sont désormais présentées à la fin de chaque chapitre pour permettre au lecteur qui vient de finir une lecture d'approfondir le sujet. Sauf exception, je ne cite que des livres ou des articles que j'ai lus. Je me suis efforcé, dans la mesure du possible, de présenter le plus possible de références en français.
- Fil rouge : avec Pablo Pernot, à l'origine de Peetic, nous avons eu l'occasion de nous exercer ensemble à Peetic au cours des Raids Agiles en Cévennes ; les exemples Peetic sont bien plus nombreux dans cette édition.
 - *Présentation du sujet* : <http://www.areyouagile.com/2012/11/peetic/>
 - *Matériel en ligne* : <https://github.com/pablopernot/peetic>

Les exemples fournis dans le livre pourront ainsi être commentés et complétés en ligne, et être présentés avec des points de vue différents.

- Le format des chapitres a été enrichi, avec un paragraphe « Sur le terrain » qui présente des cas pratiques et un tableau « Bien commencer ».
- Un glossaire explique les termes Scrum.
- Et enfin de nouveaux dessins et schémas, un quiz actualisé et des nouveaux compléments en ligne (www.aubryconseil.com).

Remerciements

Je me suis appuyé sur des relecteurs nombreux et compétents qui ont fait beaucoup pour la qualité de cet ouvrage. Cette fois, j'en ai eu de vraiment exceptionnels que je remercie du fond du cœur :

- Stéphane LANGLOIS, souvent mon premier lecteur, avec qui j'ai eu, par chapitre, environ une heure de conversation (oui, pour chaque chapitre !). Il m'a, en particulier, aidé à avoir un ton moins péremptoire et un style plus fluide.
- Alexandre BOUTIN, relecteur depuis la première édition, m'a poussé à ne pas affirmer des choses sans preuve et à mieux expliquer mes idées.
- Stéphane BÉDON-ROUANET, un lecteur extrême que je n'ai pas encore rencontré, m'a, entre autres, appris comment bien placer les virgules.
- Jacques COUVREUR venu tout spécialement de Genève à Toulouse pour m'écouter lire à voix haute quelques chapitres m'a apporté un feedback précieux avec nos conversations après ma lecture.

Merci à Nicolas DEVERGE, Laurent MEURISSE, Yannick AMEUR et Romain COUTURIER qui m'ont relu quelques chapitres, chacun dans son style particulier.

Je remercie Thierry COURTIADE qui m'a apporté pour quelques-uns des derniers chapitres un retour différent, de quelqu'un qui n'est pas un spécialiste de l'agilité. J'en profite pour remercier aussi Thierry de m'avoir dit, en juin 2009, au cours d'une randonnée vers l'étang du Laurenti, que son frère avait un bon coup de crayon.

Les dessins de Patrice COURTIADE apportent depuis la première édition leur humour décalé. Il y en a maintenant une cinquantaine, avec les nouveaux ajoutés dans cette édition quatre. Un grand merci à Patrice.

Merci à Amanda MARTINEZ qui a contribué au chapitre « Découvrir le produit ».

Je remercie bien sincèrement toutes les personnes que j'ai rencontrées lors de mes formations et interventions sur les projets, leurs retours et leurs encouragements m'ont été précieux.

Je suis très reconnaissant à Pablo PERNOT d'avoir ciselé la si flamboyante préface de cette quatrième édition.

Merci à Ruth pour son soutien sans faille au cours des nombreuses journées, soirées et week-ends que j'ai passés à écrire et réécrire ce livre.

Je termine par une dédicace spéciale à Jean-Luc MAZÉ. En septembre 2013, il a publié un commentaire sur la page Amazon de mon livre. Un commentaire positif, mais dont le titre était « Bien sûr, il y a mieux... mais en anglais ». Je crois que c'est cela qui a déclenché en moi l'idée de la possibilité d'une édition quatre. Il y a sans doute mieux en anglais, mais en tout cas j'ai fait de mon mieux pour offrir, en français, le meilleur de Scrum à mes lecteurs.

Claude AUBRY
Boncourt sur Meuse, le 30 juillet 2015.

Scrum dans le mouvement agile

En 1996, j'étais consultant en génie logiciel. Twitter n'existe pas encore, alors pour faire de la veille technologique, je lisais les comptes-rendus des conférences de l'époque. La tendance était alors l'orienté objet ; parmi les nombreuses conférences sur le sujet, il y avait OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications*). C'est en parcourant ses *Proceedings* que je suis tombé pour la première fois sur Scrum. Signé de Ken Schwaber, un article^[1], présentait Scrum comme un processus empirique pour le développement de produits complexes.

Sa lecture m'avait beaucoup intrigué. Bien que l'article fasse beaucoup de références à l'orienté objet (probablement pour être retenu à la conférence OOPSLA...), c'était en rupture avec l'état d'esprit de l'époque, et aussi avec mes centres d'intérêt qui étaient l'architecture de logiciel et la modélisation.

J'avais apprécié l'insistance sur l'équipe, et la référence au rugby m'avait attiré et intrigué. J'avais été sensible aux idées développées.

J'étais quand même loin de m'imaginer que vingt ans plus tard Scrum serait aussi populaire et, incroyable, subirait des critiques pour être perçu comme la pensée dominante.

Avant de s'intéresser en détail à ce que Scrum est devenu aujourd'hui, voyons rapidement de quoi il s'agit.

1.1. Premiers pas avec Scrum

1.1.1. Un cadre léger

Dans l'article de 1996, Schwaber parlait volontiers de processus et de méthodologie. Ensuite, Scrum a été plus souvent qualifié de méthode. Cette difficulté à classer Scrum a persisté un certain temps.

Puis Ken Schwaber et Jeff Sutherland, l'autre co-fondateur, ont défini Scrum comme un cadre de processus (*process framework*).

Scrum n'est pas un processus complet (et encore moins une méthodologie), c'est un cadre de processus.

Un processus définit une façon de travailler, un cadre de processus se contente de délimiter, de « cadrer ».

Le cadre Scrum est léger, n'imposant que peu de chose :

- les sprints et leurs événements,
- une équipe avec trois rôles,

- un backlog contenant le travail à faire.

S'il n'est pas facile de classifier Scrum, il est bien plus simple d'en expliquer la mécanique de mise en œuvre.

1.1.2. Scrum en bref

Scrum aide les gens à améliorer leur façon de travailler.

Scrum s'applique en particulier au développement de produits (ou de services ou d'applications ou de systèmes) :

- Les gens travaillent en équipe, bien définie.
- Le développement est rythmé par une série d'itérations courtes qui sont appelées des sprints.
- Les fonctions du produit sont collectées dans le *backlog*.
- Le contenu d'un sprint est défini à partir du backlog, en tenant compte des priorités et de la capacité de l'équipe. À partir de ce contenu, l'équipe identifie les travaux nécessaires et s'engage sur l'objectif du sprint.
- Pendant un sprint, des points de synchronisation sont effectués tous les jours. Cette inspection quotidienne permet d'appliquer, en équipe, des ajustements pour assurer le succès du sprint.
- À la fin de chaque sprint, l'équipe présente l'incrément qu'elle a ajouté au produit pendant le sprint. Son évaluation et le feedback récolté permettent de faire évoluer la définition du produit en cours de réalisation. L'amélioration porte aussi sur la façon de travailler en équipe.

1.1.3. Origines

Scrum n'est pas un acronyme. Le mot Scrum vient du rugby : en anglais, *scrum* signifie mêlée.

C'est pourquoi on n'écrit pas SCRUM. C'est pourquoi on prononce « screum », et non « scrume » ni « scroum ». Il suffit de regarder un match de rugby avec un arbitre anglophone pour l'entendre.

Au rugby, une mêlée est sifflée par l'arbitre quand une règle n'a pas été respectée. Par exemple :

- une équipe a fait un en-avant,
- le ballon est parti directement en touche sur un renvoi.

La mêlée permet de repartir sur des bases solides, avec une poussée synchronisée de tout le pack. C'est la quintessence de l'effort collectif que Scrum prend comme exemple de la meilleure façon d'attaquer la complexité.

Lors de ma première présentation de Scrum dans une conférence, je m'étais beaucoup appuyé sur le lien avec le rugby, voici un extrait :

« Scrum utilise les valeurs et l'esprit du rugby et les adapte aux projets de développement. Comme le pack lors d'une mêlée au rugby, l'équipe chargée du développement travaille de façon collective, soudée vers un objectif précis. Comme un demi de mêlée, le ScrumMaster aiguillonne les membres de l'équipe, les repositionne dans la bonne direction et donne le tempo pour assurer la réussite du projet. »



Fig. 1.1 La mêlée

Pourquoi cette référence au rugby ?

Les deux fondateurs de Scrum sont américains. Or les Américains ne font pas de rugby avec des mêlées. Alors ?

La référence vient d'un article de 1986 « *The New New Product Development Game* », rédigé par des Japonais, qui eux aiment le rugby et ses mêlées. Il montre comment des sociétés (Honda, Canon, NEC et Fuji-Xerox) sont performantes avec une approche de co-construction pour le développement de leurs produits, en insistant sur l'importance d'équipes autonomes et autogérées. Les auteurs [Takeuchi] présentent la métaphore du rugby ainsi :

« L'approche séquentielle traditionnelle, autrement appelée “course de relais”, pour développer un produit – illustrée par le système de planification de programme en phase – peut entrer en conflit avec les objectifs de vitesse et souplesse maximum. Au lieu de cela, une approche globale comme au rugby – où une équipe essaye de parcourir la distance en étant solidaire, se passant le ballon de main en main – peut mieux servir les exigences de compétitivité. »

Jeff Sutherland s'est appuyé sur cet article pour donner son nom à Scrum.

1.1.4. En quoi Scrum est-il fondamentalement différent ?

Scrum se différencie nettement des approches traditionnelles. Voici, à mon avis, les six nouveautés fondamentales.

1. Approche empirique

Scrum a son origine dans la théorie du contrôle empirique des systèmes complexes^[2].

Dans les domaines complexes, il n'est pas possible de prévoir à l'avance ce qui sera réellement obtenu à la fin : les spécifications détaillées, les plans détaillés faits à l'avance sont donc inadaptés. L'approche empirique s'appuie sur des cycles courts avec des rétroactions fréquentes pour avancer vers une solution inconnue au départ. L'empirisme dans Scrum est appelé « inspection & adaptation ».

2. Rythme

Scrum utilise des blocs de temps fixes pour créer de la régularité. Le cœur du rythme de Scrum est le sprint.

La nouveauté est que la fin du sprint n'est pas décidée quand un travail est achevé, mais fixée à l'avance et jamais repoussée.

3. Priorité

Pendant un sprint, le travail de l'équipe porte sur les choses qui apportent le plus de valeur. On évite de perdre du temps sur des choses sans valeur immédiate. On évite de commencer plein de travaux en même temps.

« Arrêter de commencer, commencer à finir. »

4. Autogestion

L'équipe a le pouvoir et l'autorité pour organiser son travail en fonction de l'objectif. Cela donne plus de responsabilité aux personnes impliquées et leur permet de mieux s'épanouir dans le travail.

5. Transparence

Le suivi est fait par l'équipe elle-même ; il est visible et reste simple afin qu'il soit facilement compréhensible par tout le monde. Un bon moyen pour y parvenir est de pratiquer systématiquement le management visuel.

6. Émergence

On laisse du temps à l'équipe pour favoriser l'émergence de nouvelles idées pour le produit, pour la conception et aussi pour améliorer la façon de travailler ensemble.

1.1.5. Où trouver la référence de Scrum ?

Pendant longtemps, il n'y a pas eu de livre sur Scrum. Ken Schwaber en a ensuite publié plusieurs, mais qui n'avaient pas le statut de référence. Finalement, Jeff Sutherland et Ken Schwaber ont publié un document d'une quinzaine de pages.

La référence est le *Guide Scrum*, traduit en français [Sutherland, *Scrum Guide*], dont la dernière version date de juillet 2013.

Évidemment, en 15 pages, il ne donne que les règles du jeu, présentant Scrum de façon succincte. Mon livre existe pour aider les gens à aller plus loin, tout en étant, me semble-t-il, en bonne adéquation avec le *Guide Scrum*.

Ce guide positionne Scrum comme agnostique du domaine même si, encore aujourd'hui, la plupart des expériences tournent autour de l'informatique.

1.2. Le mouvement agile

Aux débuts de Scrum, on ne parlait pas encore de méthode agile.

1.2.1. Manifeste agile

Le terme « agile », maintenant associé à Scrum, est apparu dans le domaine du logiciel avec le *Manifeste agile*^[3].

Publié au début de l'année 2001 et inchangé depuis, ce manifeste définit une attitude de réaction par rapport à des processus lourds et bureaucratiques, en vogue à l'époque (et parfois encore aujourd'hui).

Ce positionnement relatif se reflète dans la formulation « *nous privilégions les notions de gauche par rapport à celles de droite* » :

- Les gens et leurs interactions sont plus importants que les processus et les outils.
- Un logiciel qui fonctionne prime sur de la documentation exhaustive.
- La collaboration avec les clients est préférable à la négociation contractuelle.
- La réponse au changement passe avant le suivi d'un plan.

Le *Manifeste agile* représentait à l'époque de sa publication une réaction radicale à la tendance dominante, pour promouvoir des processus plus légers.

Il inclut, en plus des quatre valeurs comparées, douze principes ; voici le premier, absolument essentiel :

Satisfaire le client en livrant tôt et régulièrement des logiciels utiles qui offrent une véritable valeur ajoutée.

Scrum se range sous la bannière du *Manifeste agile*. Ses deux fondateurs font d'ailleurs partie des 17 signataires.

Le *Manifeste* définit des valeurs et des principes qui sont universels, mais la façon de les mettre en œuvre sur des projets varie. Cette application se fait par l'intermédiaire de ce qu'on appelle les **pratiques**.

Scrum impose des pratiques bien définies, que nous détaillerons dans la suite du livre. Cependant, dans le grand corpus de l'agilité, on trouve bien d'autres pratiques, d'origine différente.

1.2.2. Pratiques agiles

Une **pratique** est une approche concrète et éprouvée qui permet de résoudre un ou plusieurs problèmes courants ou d'améliorer la façon de travailler lors d'un développement.

Des **pratiques** maintenant qualifiées d'**agiles** existaient avant le *Manifeste agile* et, pour certaines, avant les premières méthodes agiles.

Par exemple, depuis le début des années 1980, la communauté du génie logiciel recommande de faire des cycles de développement courts (ce qu'on appelle le

développement itératif).

D'autres pratiques sont apparues avec les méthodes agiles et sont devenues indiscutables, après avoir été éprouvées sur de nombreux projets, par exemple la tenue d'une réunion quotidienne (la mêlée).

Prises individuellement, les pratiques sont efficaces. Insérées dans le cadre cohérent d'une approche agile, elles se renforcent mutuellement. C'est le cas dans Scrum, dont le cadre est composé d'une dizaine de pratiques majeures et en induit d'autres. C'est pourquoi Scrum est souvent qualifié de « méthode agile » (comme on peut le lire dans le sous-titre de ce livre !).

1.2.3. Méthode agile

Scrum et *Extreme Programming* (XP) existaient bien avant le *Manifeste*. Dès la publication de celui-ci, on les considéra comme des méthodes agiles, avec d'autres maintenant marginales. Depuis peu, Kanban^[4] fait partie du lot des méthodes qui comptent.

Dans les enquêtes qui montrent leur diffusion et leur utilisation, année après année, c'est Scrum qui reste la plus populaire. Au point que **Scrum** et **agile** soient des mots qu'on utilise l'un pour l'autre ou qu'on accolé : des offres d'emploi ou des articles parlent d'**Agile Scrum**.

On a vu que Scrum n'était pas vraiment une méthode mais un cadre. Kanban n'est pas non plus une méthode agile, c'est une méthode d'amélioration de processus ou une méthode de gestion de flux. Finalement, il n'y a que XP qui devrait être qualifié de méthode.



Fig. 1.2 Agile avec méthode

C'est de plus en plus rare, mais cela arrive encore que des entreprises mettent au point leur propre méthode. Pourquoi pas, mais la méthode ainsi créée n'est pas Scrum.

Il y a aussi celles qui disent passer au « mode agile » pour certains développements, pour signifier plus de flexibilité. Bien sûr, Scrum ne rentre pas dans ce mode agile. Et faire du Scrum pour être à la « mode agile », ce n'est pas mieux...

Scrum est donc considéré comme une méthode agile. Et quand on parle de méthode agile, le domaine concerné reste généralement celui du développement de logiciel.

1.2.4. L'agilité

Au-delà de l'informatique

En fédérant les méthodes agiles, le *Manifeste agile* constitue l'acte de naissance du mouvement de l'agilité qui a pris de l'ampleur depuis quelques années. Scrum est maintenant diffusé dans de très nombreuses organisations : après les pionniers et les premiers adeptes, c'est la majorité des développeurs qui s'y intéresse.

Parti du développement, Scrum implique les gens des « métiers » et irrigue même au-delà de l'informatique :

- On trouve des utilisations de Scrum pour le marketing ou pour le commerce.
- Dans le domaine du numérique, des organisations complètes y passent.
- Des expériences ont lieu pour développer du matériel (*hardware*) et des systèmes incluant matériel et logiciel.

Marc Andressen, fondateur de Netscape, a eu cette phrase célèbre : « *Software is eating the world* ».

J'ai l'impression d'assister à la même déclinaison pour les processus : les méthodes venant du logiciel sont en train de pénétrer partout.

Comme Scrum n'est pas lié à un domaine particulier, on se dit qu'il peut être utile pour développer des produits de tout type.

Le management agile

Le management est en plein changement ; les nouvelles approches [Denning, *Radical Management*] sont totalement en phase avec les principes de l'agilité. Les approches des « entreprises libérées » présentent aussi de nombreuses similitudes avec le mouvement agile.

Cependant, toutes les initiatives se réclamant « management agile » ne sont pas issues du mouvement impulsé par le *Manifeste*, leurs points de convergence avec Scrum ne sont pas toujours évidents.

Une définition de l'agilité

Jim Highsmith, un des signataires du *Manifeste*, définit l'agilité par rapport au changement :

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

Il me paraît intéressant d'ajouter une dimension vers le client final, l'utilisateur, celui qui est impacté par le résultat. Voici ma définition :

« *L'agilité est la capacité d'une organisation à fournir tôt et souvent des services impactant ses utilisateurs, tout en s'adaptant à temps aux changements dans son environnement.* »

Dit autrement, et pour reprendre ce que dit Jeff Patton [Patton, *Story Mapping*], l'objectif, en développant des produits, est de changer le monde, ne serait-ce qu'un petit peu...

Pour cela, une nouvelle culture est nécessaire.

La culture agile

Les valeurs et les principes de l'agilité peuvent présenter un caractère indéniablement subversif pour certaines organisations (mais on sait aussi que les valeurs sont assez vite récupérées).

Il importe de tenir compte des aspects culturels dans la formation et la transition à Scrum : on ne change pas si facilement de culture.

Au-delà des idées, l'agilité, en particulier avec Scrum, véhicule un vocabulaire nouveau. En quelque sorte, le vocabulaire contribue à renforcer l'idée du changement de culture.

1.3. Scrum aujourd'hui

1.3.1. Scrum mania

Quelle est la situation de Scrum dans le paysage actuel ?

L'engouement pour Scrum (la *Scrum mania* !) a mis fin à une hypothétique rivalité entre les méthodes agiles. Les études d'opinion et les tendances des recherches sur le Web le montrent : Scrum est de loin la plus populaire dans la famille des méthodes agiles.

Dans l'enquête annuelle sur l'état de l'agilité^[5] publiée par VersionOne, Scrum reste de très loin la méthode agile la plus répandue. Presque trois quarts des répondants disent utiliser Scrum ou des variantes.

L'usage de Scrum, dans le développement de produits ou de services numériques en tout cas, est probablement devenu *mainstream*. En revanche, sa mise en œuvre avec succès en est loin.

La difficulté majeure pour les acteurs de la communauté Scrum est d'amener ses utilisateurs à en faire un bon usage, sans être dogmatique.

1.3.2. Scrum bashing

Scrum ayant pris une si grande place, il est naturel de trouver des articles critiques par ceux qui proposent d'autres approches.

Et sur le terrain ? On commence à voir des déçus qui ne retrouvent pas tous les bienfaits qu'on leur avait promis : client enchanté, retour sur investissement, moins de défauts, plus vite sur le marché, plaisir dans l'équipe, etc.

D'autres ne voient Scrum que de façon partielle :

- Des utilisateurs, brimés depuis longtemps par leur direction des systèmes d'information (DSI), découvrent que Scrum peut accueillir et même favoriser les changements, ce qui les amène à penser qu'ils peuvent tout changer tout le temps.
- Des managers se disent qu'avec l'agilité, il leur sera plus facile de demander à leurs équipes de traiter une urgence par du travail supplémentaire : « *Puisque vous êtes agiles, vous devez pouvoir faire ça !* »

La réalité peut les décevoir. Voici quelques éléments de réponse à ces critiques ou déceptions.

Scrum, pour vous ?

Scrum n'est pas la solution pour tous les types de développement. On s'aperçoit que ce n'est pas tellement le domaine ou les technologies utilisées qui sont un frein, mais la complexité et la culture.

- Si vous développez des applications qui ne sont pas complexes au sens du modèle Cynefin [Lochet, *ScrumDay 2015*], Scrum n'est peut-être pas le meilleur choix.
- Si votre organisation est clairement dans une culture du contrôle et que vous n'êtes pas en mesure de promouvoir le changement radical nécessaire, Scrum n'est pas pour vous.
- Si vous travaillez continuellement dans l'urgence et que vous n'êtes pas en capacité de réduire les perturbations, vous n'arrivez pas à focaliser l'équipe sur un sprint sans qu'elle soit perturbée. Dans ce contexte d'urgence permanente, l'usage de Scrum n'est pas le plus recommandé. Kanban est probablement plus adapté, même si, comme nous le verrons plus loin, il est aussi possible d'utiliser Scrum avec Kanban.

Scrum accueille le changement, mais ne le rend ni gratuit ni permanent. Dans un usage strict de Scrum, une équipe qui travaille ne doit pas être perturbée, le changement étant possible à la fin du sprint, pas pendant.

Le changement oui, les interruptions continues non !

Scrum a évolué et évolue toujours

Parfois des gens pensent « faire du Scrum » mais leur référence à Scrum n'est pas bonne, ce qui peut expliquer leurs difficultés. Ils ont lu un article ou un livre, ou bien assisté à une formation, il y a quelques années et n'ont pas réactualisé leurs connaissances.

Certaines pratiques qui, il y a un temps, ont été associées à Scrum ne le sont plus aujourd’hui : elles sont devenues obsolètes^[6]. Car Scrum évolue, et c'est normal pour un mouvement ouvert au changement et apôtre des retours du terrain.

Le Scrum de 1995 décrit par l'article de Schwaber n'a plus grand-chose à voir, dans les pratiques, avec la dernière version du *Guide Scrum*. Plus près de nous, la première édition de ce livre, écrite en 2009, est bien différente de celle que vous lisez.

Scrum évolue, mais est aussi mal compris : des pratiques qu'on associe volontiers à Scrum n'en ont jamais fait partie. L'exemple typique est le *planning poker*.

Shu Ha Ri

Le concept de *Shu Ha Ri* [Addiniquy, *Shu Ha Ri*] vient d'Alistair Cokburn (un signataire du *Manifeste*, celui qui a proposé le mot « agile »). C'est un modèle d'évolution issu de l'aïkido.

- Le *Shu* correspond à la phase d'apprentissage où l'élève s'entraîne à reproduire ce que fait le maître, sans chercher à modifier le mouvement.
- Le *Ha* est l'étape de perfectionnement : l'élève prend des initiatives dont il comprend les impacts et qui ne modifient pas l'objectif final du geste.
- Le *Ri* est l'atteinte de la maîtrise : l'élève est devenu un maître qui peut inventer de nouvelles formes au mouvement d'origine.

Les questions sont à poser à ceux qui ont du mal avec Scrum : ont-ils bien commencé avec le *Shu* avant de faire *Ha* ? Sont-ils bien restés dans le cadre Scrum ? L'ont-ils respecté ?

Sinon il existe le risque de dénaturer, ce n'est plus Scrum.

Pour lancer de nouveaux services axés sur les rencontres entre animaux, l'équipe Peetic a choisi Scrum.

Nous verrons dans les prochains chapitres plus en détail comment elle met en place son application. Mais en tout cas, elle est bien décidée à faire du *Shu* avant d'essayer le *Ha*.

Scrum n'est pas suffisant

Scrum n'est qu'un cadre, il ne doit pas être utilisé sans lui adjoindre des pratiques complémentaires pour le domaine visé.

Pour le développement de logiciel, les pratiques d'ingénierie venant de XP sont nécessaires en complément. Sinon, comme le dit crûment mais justement @pierreroth64 sur Twitter :

« *C'est le carnage assuré et une mauvaise pub injuste pour Scrum.* »

Nous en reparlerons dans le chapitre 13 *Contextualiser Scrum*.

1.3.3. Transcendance de Scrum

Après de nombreuses expériences au niveau des équipes, Scrum est de plus en plus pratiqué au niveau des organisations et pour des gros projets.

Le passage à l'échelon supérieur, le *Scrum de Scrums*, soulève de nombreuses questions mais amène aussi de nombreux points de rencontre avec des pratiques d'organisation venant d'autres horizons. Nous les aborderons principalement dans la deuxième partie de ce livre.

1.3.4. Éléments de langage

La première rencontre qu'on a avec une nouvelle culture se fait avec les mots utilisés. Et Scrum vient avec un vocabulaire particulier.

Pour vous aider à comprendre le sens des termes, j'ai ajouté un glossaire à la fin du livre. Beaucoup de mots nouveaux, qui sont arrivés en anglais. Dans sa diffusion en France, quelques-uns sont restés en anglais, d'autres ont été traduits.

Le vocabulaire a évolué en 10 ans et continue encore à changer.

Termes non traduits

Des mots de Scrum comme *sprint* et *feedback* sont dans le dictionnaire français depuis un bout de temps. C'est le même mot en anglais et en français.

Je n'ai pas traduit d'autres termes dans ce livre. Cela ne veut pas dire qu'il n'y a pas eu de tentatives. Pour la plupart, des traductions ont été proposées^[7] mais n'ont finalement pas été retenues par les utilisateurs.

Dans cette liste, on trouve : *backlog*, *feature*, *story*, *Product Owner*, *ScrumMaster*, *burndown*, *release*, *epic*. J'explique dans les chapitres concernés pourquoi j'ai choisi de ne pas traduire ces termes.

Cependant, on peut rencontrer ces mots traduits en français, mais seulement au Québec : le *backlog* est le carnet, le *burndown* de *release* est l'histogramme de livraison restante...

Termes traduits

D'autres termes Scrum sont utilisés avec une traduction française. Pour certains, leur usage est important, mais, malheureusement de mon point de vue, n'est pas généralisé. Dans le langage courant, et parfois même dans des écrits, on trouvera : *definition of done*, *daily scrum*, *sprint planning meeting*, *sprint review*, *timebox*, *impediment*. Je défends vigoureusement l'usage du français, avec respectivement : définition de fini, mêlée quotidienne, réunion de planification du sprint, revue du sprint, boîte de temps, obstacle.

Parfois on rencontre des locutions franglaises comme définition de *ready* ou test d'*acceptance*, au lieu de définition de prêt et test d'acceptation.

Le vocabulaire évolue avec des termes qui prennent plus d'importance et les retours du terrain. Je pense en particulier à *stakeholder* et *backlog refinement*, pour lesquels je préconise maintenant parties prenantes et affinage du backlog.

Comme le français est une langue vivante, on peut penser que certains mots de Scrum seront bientôt adoptés, comme *backlog*. Cela permettra de se fixer sur son genre, pourquoi pas féminin, alors que les avis sont partagés. Parmi les néologismes, on entend fréquemment agiliste pour parler d'un adepte des méthodes agiles, un peu moins scrummeur. Pourtant il y en a beaucoup.

Bien commencer avec Scrum

La question à se poser

Pourquoi voulons-nous utiliser Scrum ? Quels problèmes voulons-nous résoudre ?

Un mauvais signe

Ce n'est pas vraiment Scrum qu'on applique, car on pense que « Chez nous c'est différent ».

Par quoi démarrer

Le Shu.

Une lecture pour tous

Les chapitres suivants.

À retenir

Sous la bannière de l'agilité et de son *Manifeste*, Scrum fournit un cadre simple pour développer des produits. Toutefois, derrière son apparence simplicité se cache une grande puissance pour mettre en évidence le degré d'efficacité des pratiques de développement utilisées.

Et comme nous allons le voir, si le cadre est simple, la mise en œuvre demande de nombreux efforts et a des impacts sur la culture des organisations.

Références

☞ Christophe Addinquiry, *Shu Ha Ri* l'article, 2015.

<http://freethinker.addinq.uy/post/86345932597/scrum-shu-ha-ri-larticle>

☞ Steven Denning, *Radical Management*, Jossey&Bass, 2010.

☞ Nicolas Lochet, ScrumDay 2015 Keynote de Dave Snowden, Blog Xebia.

<http://blog.xebia.fr/2015/04/20/scrumday-2015-keynote-douverture-de-dave-snowden>

☞ Jeff Patton, *Story Mapping*, VF, Dunod, 2015.

☞ Jeff Sutherland et Ken Schwaber, *Scrum Guide*, 2013, VF.

<http://wiki.ayeba.fr/Guide+Scrum+2013>

☞ Hirotaka Takeuchi et Ikujiro Nonaka, *The New New Product Development Game*, VF, 1986.

http://www.fabrice-aimetti.fr/dotclear/public/mes-documents/TheNewNewProductDevelopmentGame_French.pdf

Notes

[1] Qu'on ne trouve plus en ligne dans son intégralité. On a seulement accès à la soumission : <http://www.jeffsutherland.org/oopsla/schwaber.html>.

[2] Cela peut surprendre aujourd'hui ceux qui disent « *Scrum, c'est bien mais chez nous c'est trop complexe pour l'utiliser* », mais c'est pourtant l'attaque de la complexité qui est visée par Scrum.

[3] Voir <http://www.agilemanifesto.org> et <http://www.agilemanifesto.org/iso/fr/> pour la version française (traduction « officielle » de 2010).

[4] L'apparition de Kanban dans les méthodes agiles est très récente mais en fait le *Lean Software*, d'où vient le Kanban, repose sur des bases anciennes : le système de production des usines Toyota dans les années 1950.

[5] Accessible en ligne : <http://www.versionone.com/state-of-agile-survey-results/>.

[6] Par exemple : <http://www.aubryconseil.com/post/Pratiques-Scrum-obsoletes>.

[7] Mon blog peut témoigner de ces tentatives avec propriétaire de produit, histoire utilisateur, beurdone...

Le cycle des sprints

J'ai pris part à des dizaines de projets, soit en tant que développeur, soit en tant que consultant, et il n'y en a pas deux qui se soient déroulés de la même façon, bien que certains aient suivi le même processus. Il y a une grande variation dans le déroulement temporel d'un projet, appelé **cycle de développement** (ou cycle de vie).

Un cycle est défini par des **phases** et des **jalons**. Les phases se succèdent et un jalon permet de contrôler le passage à la phase suivante : une phase a des objectifs et le jalon est là pour vérifier qu'il n'y a pas de déviation par rapport à ces objectifs.

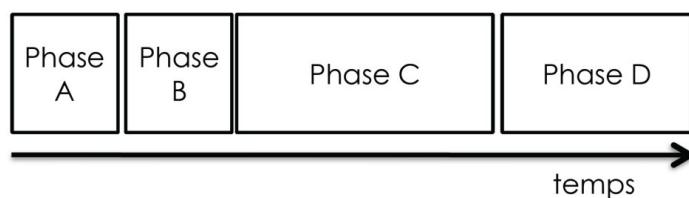


Fig. 2.1 Dans un cycle traditionnel, chaque phase est différente.

Évidemment, le cycle est influencé par le **modèle de cycle** (ou de processus) qu'on utilise. Un modèle ancien, encore répandu en France, est le **cycle en V** [Aubry, *Cycle en V*], mais les entreprises, surtout les grandes, ont souvent défini leur propre modèle de cycle.

Dans certaines entreprises, l'application du modèle est fortement recommandée tandis que dans d'autres l'équipe a plus de latitude.

Bien souvent, et quel que soit le degré de recommandation, j'ai constaté un grand écart entre le modèle et sa mise en œuvre sur les projets.

Plusieurs raisons l'expliquent :

- **Le modèle est bien souvent trop théorique**, élaboré par des « méthodologistes » éloignés des réalités, et s'avère inapplicable sur le terrain.
- **Les contrôles effectués lors des jalons** sont inopérants, parce qu'ils portent souvent sur une multitude de documents, certains faisant plusieurs centaines de pages.
- Les jalons étant franchis sans difficulté, **l'équipe accumule les travaux non faits** des phases précédentes.

Scrum possède un modèle de cycle de développement basé sur une phase qui se répète plusieurs fois successivement : le **sprint**.

Dans le langage Scrum, un sprint est un bloc de temps fixé aboutissant à créer un incrément du produit potentiellement livrable. Tous les sprints se déroulent selon le même schéma.

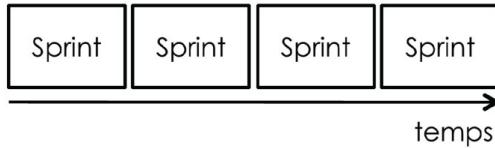


Fig. 2.2 Avec Scrum, le même schéma se répète à chaque sprint.

C'est une différence majeure avec les approches traditionnelles, pour lesquelles chaque phase impose des travaux de nature différente. Une autre différence vient du fait que Scrum n'est qu'un cadre : il ne définit pas les activités de chaque sprint, c'est l'équipe qui en a la responsabilité.

Ce cycle de développement est le sujet du chapitre ; je vais mettre l'accent sur un concept important, le rythme, donné par les sprints.

2.1. Approche itérative et incrémentale

2.1.1. Incrémentation et itération

Scrum se base sur une approche « itérative et incrémentale » pour le développement d'un produit.

Incrémental

Le terme « incrémental » met en évidence l'accroissement du produit obtenu à la fin de chaque sprint. Un processus incrémental permet de construire un produit morceau par morceau, chaque nouvelle partie venant s'ajouter à l'existant.

Pour l'écriture de ce livre, j'ai utilisé une approche incrémentale : j'ai fait un plan initial puis j'ai rédigé chapitre par chapitre, sans d'ailleurs respecter l'ordre du plan.

La pratique d'un cycle incrémental est répandue dans les développements de logiciel ; on parle souvent de lots pour les incréments qui font l'objet d'un contrat.

Itératif

Itérer est l'action de répéter. Dans le calcul scientifique, l'itération est un procédé de calcul répétitif qui permet, par exemple, de trouver la racine d'un nombre ou d'une équation, par approximations successives.

Dans le développement de logiciel, le terme **itération** est utilisé pour désigner une période de temps dans laquelle sont effectuées des activités qui seront répétées dans les itérations suivantes. Ce terme est souvent associé à processus.

Un processus itératif permet de revenir sur ce qui a été fait, dans le but de l'améliorer ou de le compléter. Cela part de l'idée qu'il est difficile, voire impossible, de faire bien la première fois. Le feedback collecté sur le résultat d'une itération permet de faire des améliorations dans la suivante. On cesse d'itérer quand le résultat obtenu est jugé satisfaisant.

Pour l'écriture de ce livre, j'ai utilisé une approche itérative : j'ai diffusé le premier jet à des lecteurs et grâce à leur feedback, j'ai produit une nouvelle version améliorée.

Itératif et incrémental

Scrum combine les deux approches avec la notion de sprint :

- à l'issue du sprint, il y a un incrément de produit qui est réalisé ;
- le feedback sollicité sur cet incrément permet d'ajuster la cible du produit dans un prochain sprint.

En résumé, un sprint est une itération qui produit un nouvel incrément (incrémental) et peut aussi enrichir l'incrément obtenu dans un sprint précédent (itératif).

Pour l'écriture de ce livre, j'ai utilisé une approche itérative et incrémentale : en fait, je n'ai pas diffusé le premier jet de tout le livre à mes lecteurs, mais je l'ai fait chapitre par chapitre. Pendant un sprint je travaillais sur la première version d'un chapitre en même temps que sur la révision d'un chapitre suite aux retours d'un ou plusieurs lecteurs.

Les organisations qui développent du logiciel en passant par la production d'une maquette, celles qui procèdent par lots, celles qui produisent une ou plusieurs versions intermédiaires disent volontiers qu'elles appliquent un processus *itératif et incrémental*.

Cycle agile

Quelles sont les caractéristiques de Scrum, en plus d'être itératif et incrémental, qui justifient le qualificatif d'agile, à propos de son cycle ?

- **Des itérations courtes** : les sprints durent d'une semaine à un mois.
- **Une séquence stricte** : les sprints ne se chevauchent pas.
- **Un rythme régulier** : les sprints ont la même durée.

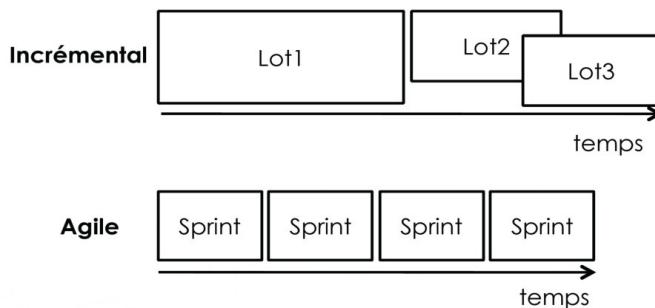


Fig. 2.3 Différences entre incrémental par lots et agile

2.1.2. Bloc de temps

Les sprints ont tous la même durée : Scrum s'appuie sur la notion de bloc de temps (*timebox*).

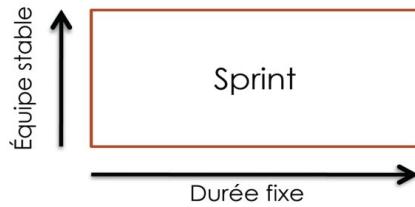


Fig. 2.4 Le bloc de temps

Pas de sprint extensible

Pour le sprint, la notion de bloc de temps s'applique de la façon suivante : on ne change pas la date de fin une fois que le sprint a commencé. Même si on n'a pas fini tout ce qu'on voulait faire !

Pourquoi ? Cela permet d'éviter le syndrome du presque fini (« *attends un peu, j'ai presque fini* ») qui peut repousser plusieurs fois la date de fin.

Je me souviens, de l'époque avant Scrum, d'équipes négociant la date d'un jalon. Ces équipes pensaient n'avoir qu'un tout petit peu de retard et demandaient à repousser la date d'une revue de deux ou trois jours. Si le délai était accordé, on constatait souvent qu'il aurait fallu encore un peu plus de temps pour finir...

La notion de bloc de temps évite cette dérive et supprime la notion de retard : à la date prévue, on fait une inspection objective de l'avancement, puis on ajuste en conséquence le contenu des prochains sprints.

Rythme régulier

La durée des sprints est toujours la même. Cela donne un rythme à l'équipe ; elle va s'habituer à produire régulièrement de la valeur.

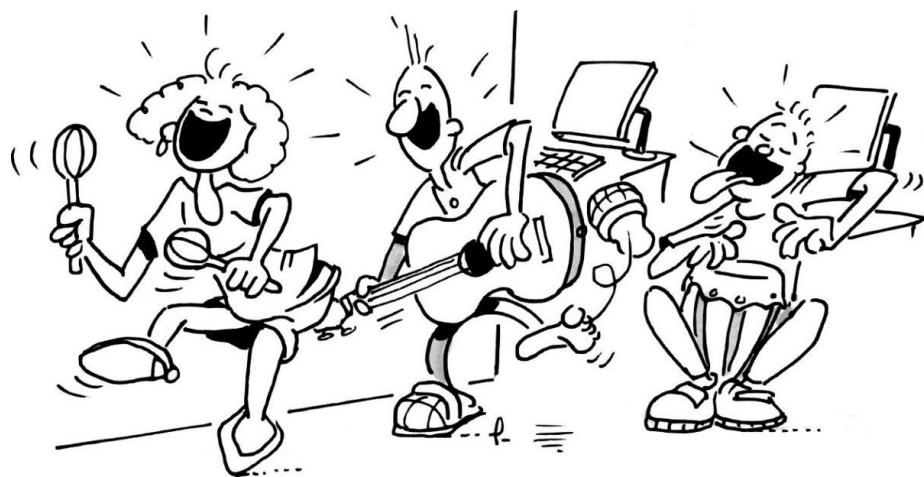


Fig. 2.5 Une équipe en cadence.

Une cadence raisonnable permet d'éviter des situations de sur-régime, que l'équipe ne pourra pas tenir bien longtemps. Pousser une équipe à travailler au-delà de son régime de croisière a des effets de bord négatifs sur la qualité de son travail : le nombre de défauts augmente, la motivation diminue, les pratiques d'ingénierie sont négligées... La dette technique pointe son nez.

Le rythme régulier présente un autre avantage : comme on connaît la date de fin de sprint à l'avance, les revues sont plus faciles à organiser : les intervenants peuvent planifier leur participation longtemps à l'avance.

Budget fixé

Le bloc de temps délimite la quantité de travail faite pendant le sprint. Il est le même pour tous les sprints : il dépend de la durée du sprint et de la taille de l'équipe, qui sont fixes toutes les deux.

Pour le développement de logiciel, le coût est déduit du temps passé, ayant pour conséquence que le budget moyen d'un sprint est fixe.

En effet, comme Scrum recommande que la composition de l'équipe reste stable, cela permet d'avoir un budget équivalent pour tous les sprints.

2.1.3. Durée du sprint

Aux débuts de Scrum, la règle était de faire des sprints d'un mois, sans variation. Aujourd'hui, on constate une tendance à faire des sprints plus courts, de deux ou trois semaines : en effet, pour le développement de logiciel, les pratiques d'ingénierie, comme l'intégration continue et les outils associés, permettent de produire des versions partielles plus fréquemment.

La durée d'un sprint est un multiple du nombre de semaines, on ne fait pas des sprints de 13 jours ou de 17 jours.

La question se pose donc de la durée des sprints pour une équipe qui démarre. Il n'y a pas de réponse universelle, chaque contexte étant différent. S'il existe un consensus pour préconiser que la durée des sprints soit fixe, il y a des variations sur cette durée.

Une enquête faite en mai 2015 sur mon blog montre que un peu moins de la moitié (44 %) des équipes fait des sprints de deux semaines et pour presqu'un quart (24 %), c'est trois semaines.

Quelques critères à examiner pour définir la bonne durée :

- **L'implication des clients et utilisateurs** – Il faut tenir compte de leur disponibilité à utiliser les versions à la fin de chaque sprint.
- **Le coût supplémentaire engendré par le sprint** – Un sprint ajoute un peu de travail supplémentaire pour produire un résultat, faire les tests manuels, préparer la démonstration pour la revue de sprint, etc.
- **La fréquence des changements** – Un sprint commencé ne doit pas être perturbé, ce qui pousse à avoir des sprints courts pour répondre plus vite aux changements.

2.2. Cycle de développement Scrum

2.2.1. L'aspect temporel

Dans un processus de développement, on distingue les jalons majeurs des jalons mineurs. Avec Scrum le jalon mineur est la fin du sprint, et le jalon majeur est la fin de la release.

Le mot *release* est d'abord compris comme « version ». Il est synonyme de mise sur le marché, avec un numéro permettant d'identifier la version. Exemple : Firefox 37.

Ce n'est pas ce sens qui est utilisé dans mon livre. J'appelle release une période de temps.

Une release est période de temps constituée de sprints.

Historiquement, cette période, le **cycle de production d'une release**, se terminait au moment de la sortie de la version. Cependant, ce n'est plus toujours le cas : il est possible de livrer une version à un rythme différent. C'est d'ailleurs une tendance dans le mouvement agile de fournir plus souvent de la valeur, à travers la version.

J'ai conservé le mot *release* car le plus souvent la fin de la release et la sortie de la « version » coïncideront (mais attention, pas toujours).

La production d'une version ne se fait pas uniquement en fin de release. On peut déployer à la fin de chaque sprint, voire pendant le sprint. On peut aussi ne déployer qu'après plusieurs releases.

Une release est donc une série de sprints. Elle se termine soit quand la décision est prise de l'arrêter, soit à une date fixée à l'avance.

Je recommande la release de durée fixe, comme le sprint.

La tendance est à raccourcir les durées : une release dure environ trois ou quatre mois, avec des sprints de deux ou trois semaines. Cela permet de dérouler de quatre à six sprints dans une release. Nous verrons tout l'intérêt de cette notion dans le chapitre 16 *Planifier la release*.

L'équipe Peetic a choisi de faire des sprints de deux semaines et des releases de trois mois. Chaque release comportera cinq sprints.

2.2.2. Activités et cycle de développement

Un **cyclde de développement** se présente comme un enchaînement de phases pendant lesquelles on effectue des activités. Pour un développement de logiciel, les activités sont généralement : Spécification, Architecture (conception), Codage, Test (d'intégration et de recette). Pour simplifier, je vais utiliser les lettres S, A, C et T pour désigner ces activités dans les schémas.

Avec Scrum, chacun des sprints se termine avec un produit qui fonctionne, ce qui nécessite du travail dans toutes les activités de développement. Les activités se déroulent en parallèle pendant le sprint.

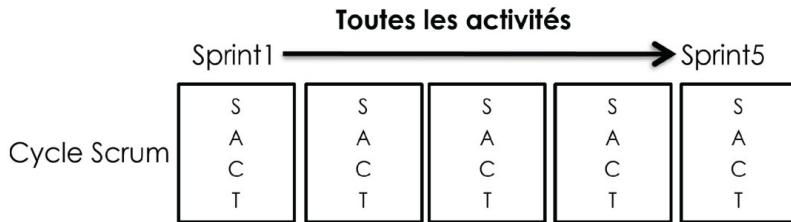


Fig. 2.6 Des sprints et leurs activités en parallèle

À l'autre extrême, un processus dit séquentiel déroule les activités en séquence, avec une activité par phase.

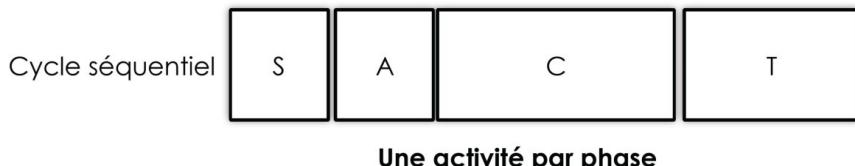


Fig. 2.7 Phases avec des activités séquentielles

Avec un processus à activités séquentielles, une **phase** est définie avec un objectif exprimé par une liste de documents à produire et dure longtemps (quelques mois). Sa durée est variable : l'équipe s'arrête lorsque les objectifs sont atteints. Au début, et même assez tard dans le développement, le résultat porte uniquement sur des documents car le logiciel testé n'est obtenu qu'à la fin.

Suivre au pied de la lettre cette approche revient à avoir :

- 100 % de la spécification détaillée avant de commencer la conception.
- 100 % de la conception avant de commencer le code.
- 100 % du code pour commencer les tests.

C'est un modèle idéal mais utopique : dans la réalité, on revient toujours sur les activités des phases précédentes, et en particulier dans la phase de test, on revient sur les spécifications, la conception et surtout le code.

Ce décalage entre le modèle et la réalité est une des raisons des échecs des projets et de leur qualité médiocre.

L'approche agile est plus pragmatique : les activités de spécification et de conception sont continues. De nouvelles fonctions et l'architecture vont continuer à émerger pendant la release, les tests sont pratiqués dès le premier sprint.

À l'autre extrême de l'approche séquentielle, il existe des développements qui sont menés de façon chaotique, sans suivre de processus ou en suivant « *la méthode à l'arrache* ». Le plus souvent, il n'y a pas d'activité de spécification ni de conception. L'équipe se lance directement dans le codage, qui est suivi d'une longue période de test et de corrections de bugs.

Scrum est bien différent de cette façon de faire : la qualité n'est pas négligée et les tests font partie des activités de chaque sprint.

2.2.3. Les trois périodes d'une release

La notion de cycle de vie est peu mise en évidence dans Scrum. Le premier article de 1995 évoquait trois phases : une première *Planning & Architecture*, la deuxième constituée des sprints et une dernière appelée *Closure*. Ces notions sont aujourd’hui abandonnées.

Néanmoins, il y a bien trois périodes distinctes dans une release :

- La période centrale, la principale, celle des sprints.
- La période avant le premier sprint.
- Une période après le dernier sprint et avant la fin de la release.

Sprint zéro

Une release commence par des travaux particuliers avant de lancer les sprints successifs, comme constituer l’équipe, définir la vision, produire un backlog initial et élaborer une première planification de la release. Si la release en question est la première dans la vie du produit, il conviendra de faire des travaux de définition de produit et d’architecture avant de lancer les sprints.

Cette période de temps en début de release est appelée le « sprint zéro ». Ce terme est trompeur parce que cette période n'est pas un sprint comme les autres : sa durée est différente, les tâches qu'on effectue sont spécifiques, il n'y a pas le cérémonial habituel des sprints, on ne produit pas un incrément de produit à la fin, on ne mesure pas de vitesse, etc.

Bien que sprint zéro soit une appellation discutable, elle est devenue la norme. Au-delà du vocabulaire, il faut bien comprendre qu'il s'agit d'une phase différente de la série des sprints qui va suivre.

Le premier sprint ne doit pas démarrer trop longtemps après le début de la release : il ne s'agit pas d'y recaser les activités de spécification et de conception détaillées. Sa durée dépend du contexte : il faut évidemment plus de temps dans le cas d'une première release d'un nouveau produit s'appuyant sur une nouvelle technologie que pour sa énième release.

Dans la plupart des situations, on déroule simplement trois activités :

- mettre en place l'équipe, son environnement de développement et l'infrastructure ;
- préparer le backlog initial ;
- prouver rapidement que l'équipe sait produire.

Fin de release

Finir le travail le mieux possible pendant les sprints évite de le faire plus tard. Si l'état du produit en fin de sprint est équivalent à l'état attendu en fin de release, la fin du dernier sprint coïncidera avec la fin de la release.

Dans certaines situations, il faut du temps entre la fin du dernier sprint et la livraison de la « version » pour faire des travaux nécessaires avant la mise en production. Ces travaux varient selon le type de déploiement : mise en production à chaud, packaging du produit,

mise à disposition en ligne, etc. Cette période a été appelée « sprint de stabilisation ». Ce n'était pas une bonne idée, cela laissait entendre que le logiciel n'était pas stable auparavant. Or, il devrait l'être à la fin de chaque sprint.

Après le dernier sprint et avant de commencer la release suivante, quelques jours peuvent être consacrés à des activités différentes : une rétrospective de release, la célébration des succès collectifs, etc.

2.2.4. Utiliser le résultat d'un sprint

En fin de sprint, on peut identifier trois usages de la version produite, pour un développement de logiciel.

Scrum appelle le résultat du sprint un incrément de produit.

- **Utilisation interne pour apprendre** – L'incrément n'est utilisé qu'au sein de l'équipe de développement. Il a pour objectif de minimiser les risques liés à la technologie et à la capacité de l'équipe à intégrer différents composants. Il apporte de la valeur, car il permet d'apprendre sur le produit ou le service. C'est ce qu'on appelle de la valeur de connaissance. Cela est particulièrement fréquent lors des premiers sprints d'une release.
- **Obtention de feedback de parties prenantes** – L'incrément est fourni à des parties prenantes sélectionnées. Cela leur donne la possibilité de prendre en main le produit, et permet ainsi de réduire les risques portant sur l'ergonomie. Ces utilisateurs pourront évaluer la facilité d'utilisation des fonctionnalités et en proposer de nouvelles. Ces retours iront alimenter le backlog pour prise en compte ultérieure. C'est le cas « standard » de fin de sprint.
- **Mise en production** – L'incrément est mis en production pour être « exploité » par ses utilisateurs finaux. C'est évidemment ce qu'il faut viser, puisque chaque nouveau sprint apporte de la valeur. Autant déployer le plus tôt possible.

Traditionnellement, les activités de mise en production étaient réalisées après le développement et par d'autres personnes. La tendance, poussée par le mouvement *DevOps* [Martignole, Devops], est à intégrer les opérations dans le développement. Cela contribue, de plus en plus souvent, à déployer à chaque sprint, voire pendant un sprint.

2.3. Les sprints et releases sur le terrain

2.3.1. Tester n'est pas une phase

Dans les cycles de vie traditionnels, le test est une phase.

Avec l'approche itérative de Scrum, le test n'est plus une phase qui se déroule après le développement. Il est intégré dans chaque sprint. Il n'est pas relégué à la fin du sprint. Dès les premiers jours du premier sprint, l'équipe commence à tester.

Cette façon de procéder permet de réduire le délai entre le moment où une erreur est introduite dans le logiciel et celui où elle est corrigée. On sait depuis longtemps que plus ce délai s'allonge, plus le coût de la correction est élevé.

2.3.2. Enchaîner les sprints

Si on se réfère à l'athlétisme, objet de la métaphore, on ne peut pas « sprinter » sans arrêt, il faut des phases de récupération.

Penser à un sprint Scrum comme une course dans laquelle l'équipe se donne à fond pour récupérer ensuite n'est pas adapté. En fait ce ne sont pas les gens qui sprintent, ce sont les éléments du backlog.

Pour les membres de l'équipe, un développement avec Scrum s'apparente plus à une course à un rythme régulier, sans pause à chaque étape. Les sprints s'enchaînent, la pause intervient seulement en fin de release.

Si des membres de l'équipe font remonter qu'ils souhaitent des jours de récupération entre les sprints, c'est le signe d'un dysfonctionnement, et il faut y prêter la plus grande attention.

Gardons à l'esprit le principe « rythme soutenable » du *Manifeste agile*.

2.3.3. Fournir le minimum viable pour apprendre

Le *Lean Startup* [Deverge, *Lean Startup*] a popularisé l'idée de faire des cycles courts pour apprendre sur le produit. On entend parler à tout va de MVP (*Minimum Viable Product*), terme mal choisi car il ne s'agit pas d'avoir un produit, même minimal à la fin d'un cycle, mais simplement un résultat qui permet de valider des hypothèses, et donc d'apprendre.

De la même façon, il faut bien comprendre que le résultat obtenu avec Scrum à la fin d'un sprint a pour but, grâce au feedback, d'apprendre sur le futur produit. Il n'est pas utile pour cela que l'incrément de produit, résultat du sprint, soit complet ou livrable : il suffit qu'il permette d'apprendre sur le produit.

2.3.4. Nommer les releases

Donner un nom à chaque release facilite la communication avec les parties prenantes. Cela marque plus que les numéros.

Exemple : on choisit le nom d'une release à partir des albums de Peter Hammill, des sommets des Pyrénées ou des joueurs du Stade toulousain. L'équipe Peetic a choisi les Pyrénées.

Bien démarrer

La question à se poser	Est-ce que la durée de la release est annoncée aux parties prenantes, ainsi que son rythme régulier ?
De mauvais signes	<p>On en est au 18^e sprint et ça continue.</p> <p>L'équipe a l'impression d'être continuellement sous pression.</p> <p>L'équipe est censée utiliser Scrum mais le processus de l'organisation est toujours bien présent.</p>
Par quoi démarrer	Définir les durées des sprints et des releases. Pour le sprint, le choix est généralement entre deux ou trois semaines. Pour la release, partir sur trois ou quatre mois.
Une lecture pour tous	Compléments sur le développement itératif et incrémental [Cockburn]

À retenir

Avec Scrum, un produit est développé selon une approche itérative et incrémentale. Le sprint donne le rythme auquel sont produites des versions partielles et potentiellement livrables du produit. La release est la séquence de sprints qui possède aussi son rythme, de l'ordre du trimestre.

Cette approche a pour objectif de produire de la valeur rapidement : soit en apprenant, soit en déployant.

Références

- ☞ Claude Aubry, *Le cycle en V n'existe pas*, Web, 2007.
<http://www.aubryconseil.com/serie/Cycle>
- ☞ Alistair Cockburn, *Utiliser les développements itératif et incrémental ensemble*, VF, Web, 2009.
<http://www.fabrice-aimetti.fr/dotclear/index.php?post/2009/11/14/Utiliser-le-developpement-incremental-et-itératif-ensemble>
- ☞ Nicolas Deverge, *Le Lean Startup*, Web, 2015.
<http://www.aubryconseil.com/post/Le-Lean-Startup>
- ☞ Nicolas Martignole, *Devops expliqué simplement*, Web, 2010.
<http://www.touilleur-express.fr/2010/12/04/prod/>

Les gens de Scrum

Dans les éditions précédentes, je n'avais pas dédié de chapitre à l'équipe, bien que j'en parlasse dans le chapitre ScrumMaster et un peu partout ailleurs. J'ai pourtant du vécu sur le sujet : pendant de nombreuses années, j'ai fait partie d'équipes diverses et variées. Mais je considérais que la vie en équipe se pratique quotidiennement et qu'elle relève de l'expérience. Il me semblait utile d'évoquer la théorie uniquement pour les compétences des deux rôles essentiels de Scrum : le Product Owner et le ScrumMaster. Aujourd'hui, j'ai trouvé des mots pour parler des gens de Scrum.

L'importance des gens, le côté humain, c'est le principal différentiateur de Scrum et de l'agilité. Le génie logiciel, le cycle en V, UML, et même la gestion de projet et le management ne s'en préoccupent pas tant. Il est d'ailleurs frappant de constater que dans les conférences agiles, comme le ScrumDay 2015, ce sont les sujets sur les gens qui intéressent le plus les participants. C'est cohérent avec la première valeur du *Manifeste agile* :

« *Les gens et leurs interactions sont plus importants que les processus et les outils* ».

Dans certaines citations du *Manifeste* en français, *individuals* est traduit par individus. Je trouve que les gens c'est bien mieux. Individu, cela fait rapport de police, cela a donné individuel...

L'équipe, avec donc les gens qui la composent et qui sont en interactions, est au cœur de Scrum.

Dans Scrum, ce qui est le plus important, ce sont les gens.

Autour de l'équipe, il y a aussi ceux pour lesquels l'équipe produit des résultats. Scrum fait la distinction entre ceux qui sont à l'intérieur, dans l'équipe, et ceux qui sont à l'extérieur. Cela fait encore beaucoup de monde à l'extérieur, direz-vous, mais nous allons considérer les seules personnes intéressées par les résultats produits, appelées des parties prenantes (*stakeholders* en anglais).

3.1. Importance des gens

3.1.1. Changement de paradigme

Pour les entreprises qui ont bâti des hiérarchies sur plusieurs niveaux et qui ont la culture du contrôle, le changement induit par Scrum est radical : on privilégie la confiance au contrôle et on favorise l'autogestion plutôt que le pouvoir du chef.

Dans les équipes Scrum, les gens ne sont pas considérés comme des ressources mais comme des créateurs de valeur.

Le management n'est pas supprimé pour autant, mais devient l'affaire de tous. Ce nouveau paradigme n'est pas spécifique de Scrum, il entre dans un mouvement bien plus large de remise en question du taylorisme.

3.1.2. Valeurs partagées

Scrum est associé à cinq valeurs : focus, ouverture, respect, courage et engagement. Dans *Exploring Scrum*, [Rawsthorne, *Exploring Scrum*] les auteurs ajoutent la visibilité et l'humour.

J'ai du mal à percevoir la visibilité comme une valeur, par contre l'humour me va bien.

Nous verrons dans les chapitres suivants comment ces valeurs sont déclinées au cours de la vie d'une équipe. Seul le focus mérite une explication immédiate.

Le focus ou la focalisation, c'est la volonté de se concentrer sur un point. Pendant le sprint, l'équipe a le focus sur l'objectif qu'elle a défini lors de la planification.

Focus, engagement, courage, respect, ouverture et humour : ces valeurs ne sont pas l'apanage des équipes Scrum^[1]. Mais dans Scrum, elles forment un ensemble cohérent en se renforçant mutuellement.

Lors de l'analyse des problèmes rencontrés à la fin d'un sprint, on s'aperçoit que la cause profonde vient souvent d'une violation d'une valeur de l'équipe. Car il ne suffit pas de proclamer des valeurs, ce que fait toute entreprise avec son service de communication, il faut aligner ses comportements avec elles.

3.1.3. Intérieur / extérieur

Scrum présente une distinction nette entre ceux qui font et ceux pour qui c'est fait. Les gens qui font constituent l'équipe Scrum. Les autres sont appelés des *parties prenantes*. Tous doivent adhérer aux mêmes valeurs, mais les pratiques ne les concernent pas de façon identique. Les membres de l'équipe sont évidemment beaucoup plus impliqués que les parties prenantes. Mais ces dernières sont tout de même concernées par quelques pratiques.

Parties prenantes

Le développement d'un produit ou d'un service est réalisé pour avoir un impact sur le comportement de personnes, plus ou moins nombreuses.

On y trouve des utilisateurs, des clients mais aussi des sponsors, des représentants des utilisateurs, des gens du marketing ou du commerce, des managers, des personnes impliquées dans l'infrastructure, dans la qualité, etc. Des gens, quoi.

Ces gens qui sont impactés sont les parties prenantes (PP), dont certains sont des experts (EXP) qui aident l'équipe.

Équipe

L'équipe Scrum est composée des personnes qui contribuent à produire un résultat à chaque sprint.

Il y a seulement trois rôles parmi ces membres :

- Deux rôles spécifiques de Scrum, qui feront l'objet des chapitres suivants : le Product Owner (PO) et le ScrumMaster (SM).
- Le développeur (DEV) contribue au développement, au sens large. Cela concerne en fait tous les membres de l'équipe.

Le terme « développeur » est discutable, mais comme c'est celui employé dans le guide Scrum, je vais le conserver.

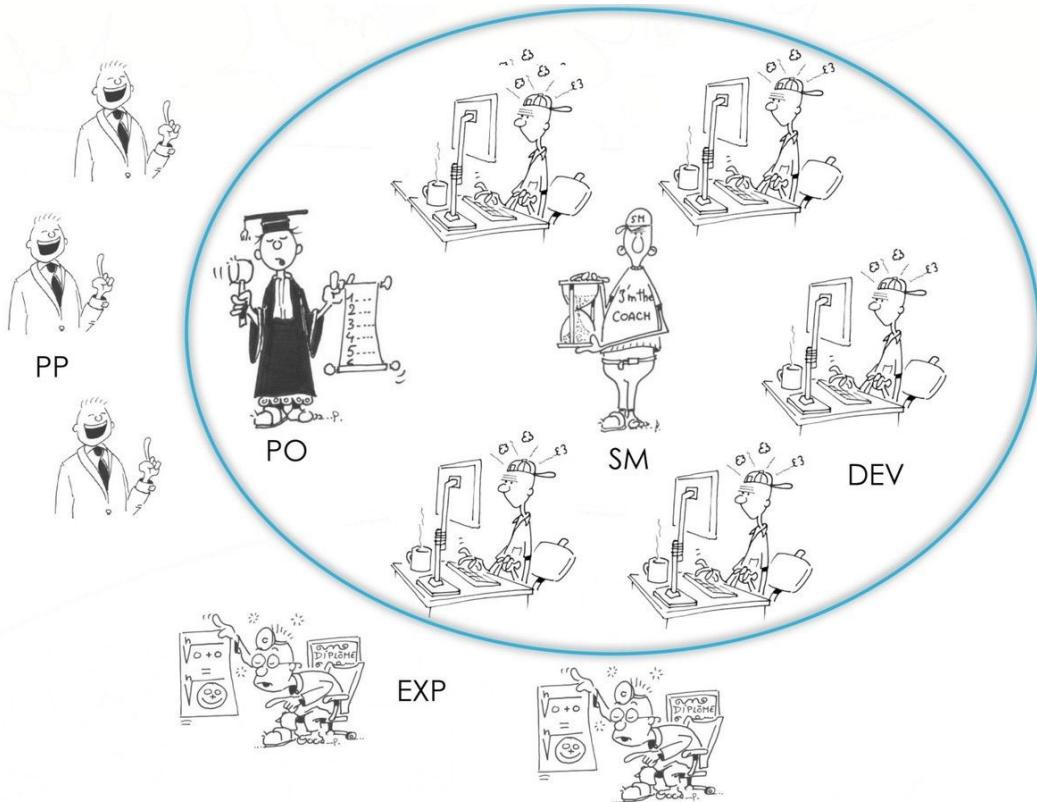


Fig. 3.1 Les rôles des gens de Scrum

3.2. L'équipe Scrum

3.2.1. Taille

Pour que le cadre Scrum reste efficace, le nombre de personnes dans une équipe doit rester limité.

Le niveau de satisfaction d'appartenance à un groupe varie selon la taille de ce groupe. Des recherches évaluent l'optimum à 7 [Pernot, *La horde agile*, p. 13].

En 2013, j'avais fait un sondage sur la taille des équipes :

<http://www.aubryconseil.com/post/Enquete-sur-la-taille-des-equipes-Scrum>

- Une seule personne, cela ne constitue pas une équipe.
- De 2 à 4, cela représente une petite équipe Scrum.
- Entre 5 et 9, on a la taille idéale^[2].
- À partir de 10, l'équipe est trop grande pour bien faire du Scrum.

Cela ne veut pas dire que Scrum n'est plus applicable, simplement il faut le faire à plusieurs équipes. C'est le Scrum à grande échelle, dont nous reparlerons plus loin.

L'équipe Peetic est composée de sept personnes, y compris le PO et le SM. Elle est entourée de parties prenantes et d'experts.

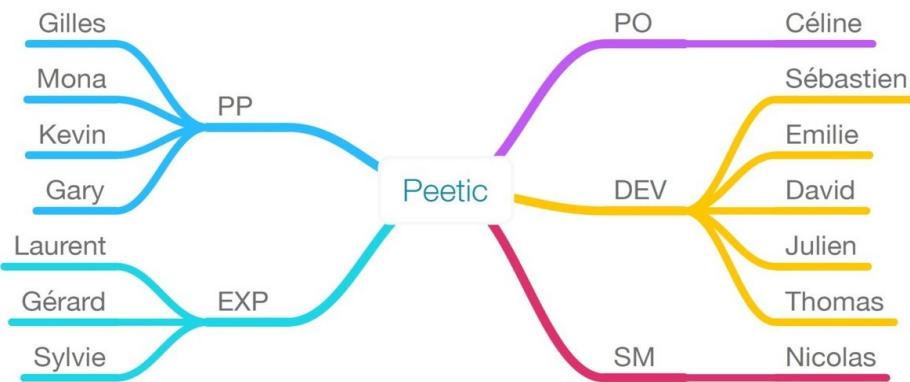


Fig. 3.2 Les gens de Peetic

3.2.2. Auto-organisation

C'est l'équipe qui réalise le produit, en développant un nouvel incrément à chaque sprint. Elle est investie du pouvoir et de l'autorité pour le faire de la façon qu'elle estimera la plus efficace : elle définit l'organisation des travaux. Chaque membre de l'équipe apporte son expertise, la synergie améliorant l'efficacité globale.

L'origine du nom Scrum, le rugby, rappelle d'ailleurs le rôle central du collectif. Prenons justement un exemple du rugby pour définir l'intelligence situationnelle, aboutissement de l'auto-organisation, avec un article de Pierre Villepreux.

« ...une distribution des joueurs les plaçant dans des situations de jeu où ils devraient pouvoir développer des compétences autres que celles de leur poste.

L'investissement de chaque joueur dans ces tâches qui ne relèvent pas du poste, se doit de répondre au contexte situationnel donc au sens attribué par toute l'équipe à cette situation.

On peut même dire quand tout est bien huilé et que chacun joue juste dans le rôle qui lui est momentanément imparti dans la situation existante que cette “ synchronisation forcément collective ” constitue alors un véritable système de jeu qualifié d’ouvert. »

Une équipe Scrum pourrait faire sien ce texte, en remplaçant « jeu » par « développement » et « ouvert » par « agile ».

3.2.3. Pluridisciplinarité

Idéalement, une équipe Scrum possède en son sein toutes les compétences nécessaires pour finir le travail dans un sprint.

Une équipe est pluridisciplinaire quand elle couvre, collectivement, toutes les disciplines requises pour développer le produit.

La pluridisciplinarité a pour objectif de permettre à l'équipe de fournir de la valeur sans dépendre d'autres équipes ou d'autres personnes. Pour l'atteindre, « développer le produit » devrait aller jusqu'à « au déploiement à ses utilisateurs ».

Selon le domaine, une équipe Scrum embarquera donc des personnes avec des compétences variées et complémentaires.

Par exemple, pour du développement de logiciel, on regroupera des compétences de définition de produit, d'expérience utilisateur (UX), d'architecture du logiciel, de développement proprement dit (le code), de tests à différents niveaux, de documentation, etc.

« Pluridisciplinaire » ne signifie pas que chaque membre de l'équipe a toutes les compétences. Cela veut dire qu'avec tous ses membres, l'équipe les possède.

Cependant, avant d'y arriver, il pourra être nécessaire de faire appel à des parties prenantes ayant une spécialité qui manque. Ces experts interviendront ponctuellement pour aider l'équipe à produire des résultats.

3.2.4. Stabilité

Une équipe Scrum est bien délimitée, on sait qui est dedans. Elle est aussi stable sur une longue durée, qui est celle de la release, au moins. Rester avec la même équipe permet d'apprendre à mieux travailler ensemble.

Le modèle de Tuckman présente les étapes de constitution d'une équipe [Messager, *Coacher une équipe agile*, p. 112] :

- création et structuration,
- confrontations et tensions,
- régulation et normalisation,
- synergie et performance,
- dissolution et séparation.

La stabilité permet de laisser du temps aux équipes pour tendre vers l'harmonisation et la performance.

L'équipe Peetic vient juste d'être créée, mais quatre développeurs Émilie, David, Julien et Thomas, ont déjà travaillé ensemble. Ils sont tous d'accord pour rester au moins une release ensemble.

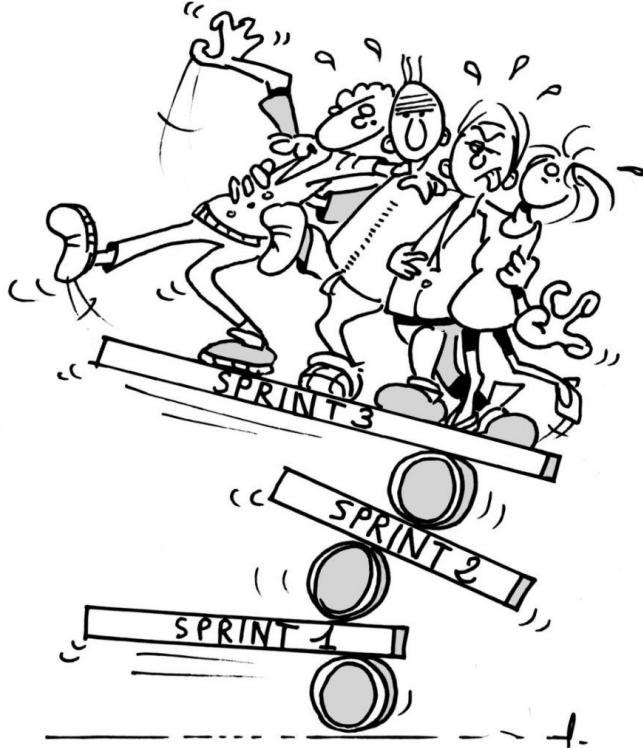


Fig. 3.3 Une équipe en recherche de stabilité

3.2.5. Identité

Une bonne équipe Scrum possède une forte identité. L'adhésion à des valeurs communes y contribue.

Une équipe pourra reprendre les valeurs de Scrum, présentées plus haut, ou définir son propre jeu de valeurs partagées, par exemple lors d'un atelier pendant le sprint zéro.

L'identité est renforcée par l'espace physique dans lequel vivent les gens. Un bureau dédié à l'équipe, avec des murs ou des cloisons, permet l'expression et le renforcement de l'identité collective. C'est pourquoi il est largement préférable que tous les membres de l'équipe vivent dans le même espace. Il est possible de déroger ponctuellement à cette unité de lieu, avec une personne ou deux en télétravail (en *remote*), mais pas d'instaurer cette distance de manière systématique, surtout si la langue et la culture ne sont pas partagées.

Une équipe distribuée géographiquement sur plusieurs continents ou plusieurs pays de culture différente, avec des membres qui ne se connaissent pas bien, n'est pas une équipe Scrum.

Le lieu idéal pour les réunions est l'espace de travail ouvert (on parle parfois de plateau projet ou de salle dédiée^[3]) dont dispose l'équipe. Du point de vue logistique, cela signifie une salle, avec les postes de travail disposés de façon à favoriser la communication, et une zone d'affichage. Un tableau accroché au mur répond à ce besoin, à condition qu'il soit suffisamment grand, visible de tous et permette un accès facile. Le tableau de l'équipe est

un outil essentiel, qui facilite le partage des expériences communes et des émotions ressenties, portant sur les trois sens : ouïe, vue, toucher.

Un tableau physique est recommandé, quels que soient les outils informatiques utilisés par ailleurs.

L'identité du groupe se construit par des réussites. Les revues de fin de sprint sont l'occasion de les présenter aux parties prenantes, puis de les célébrer ensemble.

3.3. Le développeur

Le développeur est un membre de l'équipe Scrum.

3.3.1. Responsabilité

En tant que membre d'une équipe autogérée, le développeur est responsable de ce qu'il fait devant ses pairs.

Ce qu'il fait est décidé en commun avec les autres membres, dans un engagement collectif, cela ne lui est donc pas imposé par une autorité quelconque.

Le développeur exerce sa responsabilité en contribuant du mieux qu'il peut à l'effort collectif.

Les mesures qu'on collecte en Scrum ne sont pas individuelles : pas de mesure du temps passé, ni de la performance. Cette liberté donnée aux gens implique de la transparence entre tous les membres. Le développeur se doit de remonter les obstacles qu'il rencontre et qui pourraient empêcher le groupe d'arriver à ses fins.

3.3.2. Compétences

Un développeur possède des compétences qu'il met au service de l'équipe pluridisciplinaire.

Il est compréhensible qu'il cherche à les renforcer afin de conforter sa spécialité, cependant il est souhaitable qu'il cherche aussi à les partager avec les autres membres de l'équipe. Scrum pousse à ce partage, qui présente deux bénéfices : le risque lié à une connaissance détenue par une seule personne disparaît, et chacun acquiert de nouvelles compétences.

3.3.3. Sélection

Pour constituer l'équipe, on prend en général les gens qui sont là et ça se passe bien.

Si on peut choisir, on prendra la personne qui rentre le mieux dans le cadre collectif et qui favorise le plus la pluridisciplinarité. Il est cependant préférable que ce soit une invitation à entrer dans l'équipe plutôt qu'une obligation.

Sébastien, le cinquième développeur de l'équipe Peetic a été coopté par les autres.

La plupart des gens sont heureux de faire du Scrum (si les valeurs sont respectées).

3.3.4. S'améliorer dans le rôle

Il ne suffit pas d'appartenir à la même équipe pour travailler véritablement en équipe ! On n'en est pas vraiment une si, dans le partage du travail, chacun travaille sur son morceau dans son coin.

C'est facile d'être une équipe quand on est tous ensemble dans la même activité, par exemple la mêlée, au rugby et avec Scrum. C'est un peu plus difficile quand il faut former spontanément des sous-groupes de quelques personnes pour une activité technique spécifique, comme le déblayage au rugby ou le codage en logiciel.

Des techniques pour l'organisation du travail, comme l'essaimage, et pour le développement, comme le *pair programming*, permettent de développer cette intelligence collective. Nous les présenterons dans de prochains chapitres.

D'une façon générale, Scrum favorise l'entraide entre les développeurs, leur permettant de s'améliorer continuellement.

3.3.5. Leader technique

Dans une équipe Scrum, on trouve un Product Owner, un rôle, le ScrumMaster, qui se rapproche de Process Owner, mais il n'y a pas de rôle de Technical Owner. Cependant, il n'est pas interdit d'avoir une personne qui possède le leadership technique. Cela peut favoriser la prise de décision sur des choix de conception.

Un poste de leader ou référent technique peut renforcer la pluridisciplinarité en attirant dans l'équipe une personne avec de fortes compétences techniques. Mais c'est à l'équipe de décider si elle en a besoin.

David est le développeur Peetic qui connaît le mieux Meteor, le framework choisi. Il est le référent technique pour les deux premiers sprints. Ensuite, il est prévu de tourner pour ce poste.

3.4. Les parties prenantes

Les parties prenantes (PP) sont toutes les personnes intéressées par les résultats.

3.4.1. Qui sont-elles ?

Les développeurs sont aussi des parties prenantes, mais pour simplifier nous allons considérer dans la suite que les PP sont uniquement les personnes extérieures à l'équipe.

Cela peut représenter beaucoup de monde. Pour les produits grand public, les utilisateurs auront des représentants comme parties prenantes.

On peut distinguer plusieurs catégories de PP :

- les gens du métier,

- les managers,
- les autres services et équipes impactés.

Les experts sont une catégorie spéciale de parties prenantes, que nous étudierons ci-après.

Les parties prenantes pour Peetic sont Gilles et Mona, les Peetic Holders, et Kevin le commercial.

3.4.2. Responsabilité

Une PP est moins impliquée qu'un membre de l'équipe. Elle ne réalise pas de travaux contribuant au résultat.

Néanmoins, elle se doit de participer à un événement du sprint : la revue.

Au cours de cette revue, elle est sollicitée pour donner son feedback. L'objectif clair de cette participation est la valeur ajoutée que les PP sont susceptibles d'apporter au produit. Cela correspond à de la valeur de connaissance, ce qui est appelé la « *learning value* » dans le Lean Startup.

Les parties prenantes peuvent aussi aider à éliminer un obstacle qui est hors du pouvoir de l'équipe.

Pour bien exercer leur responsabilité, il est recommandé que les parties prenantes connaissent le cadre Scrum. Il est indispensable qu'elles respectent les valeurs de l'équipe.

Par exemple, un responsable de clients qui demande une correction directement à un développeur, ou un manager qui retire quelqu'un de l'équipe pour une urgence, remet en question l'engagement de l'équipe.

3.5. Les experts

Un expert est une personne, à l'extérieur de l'équipe, qui possède des compétences nécessaires à l'équipe pour réaliser un travail d'un sprint.

3.5.1. Qui sont-ils ?

Un expert est une partie prenante qui aide l'équipe à produire un résultat. Il dispose de compétences que l'équipe ne possède pas.

À la demande de l'équipe, l'expert participe, ponctuellement, à un sprint.

Il peut s'agir d'un spécialiste du métier, c'est un **expert fonctionnel**, ou de la solution, c'est alors un **expert technique**.

Gérard a l'expérience des croquettes pour chien. Il est incontournable pour aider l'équipe Peetic sur toutes les fonctions liées à la nourriture.

3.5.2. Responsabilité

En plus de participer à la revue comme les PP, l'expert peut être présent lors d'autres événements du sprint, aux moments où il intervient.

Il ne s'engage pas systématiquement comme les membres de l'équipe. Il est souhaitable qu'il le fasse chaque fois que sa participation est requise.

Comme il est à l'extérieur, son engagement est plus problématique que celui des développeurs, ce qui est un facteur de risque dans la réussite du sprint.

3.5.3. Limiter les dépendances

L'expert peut intervenir pour plusieurs équipes. Il est généralement très demandé. Pour diminuer le risque lié à cette expertise extérieure, voici quelques pistes :

- Constituer une équipe en ayant à l'esprit de dépendre de peu d'experts, bien identifiés.
- Faire entrer dans l'équipe un expert dont on a besoin régulièrement.
- Anticiper le besoin avant le début du sprint, c'est-à-dire ne pas commencer un travail sans s'assurer que l'expert requis est disponible.
- Faire en sorte d'apprendre de l'expert pour pouvoir s'en passer.

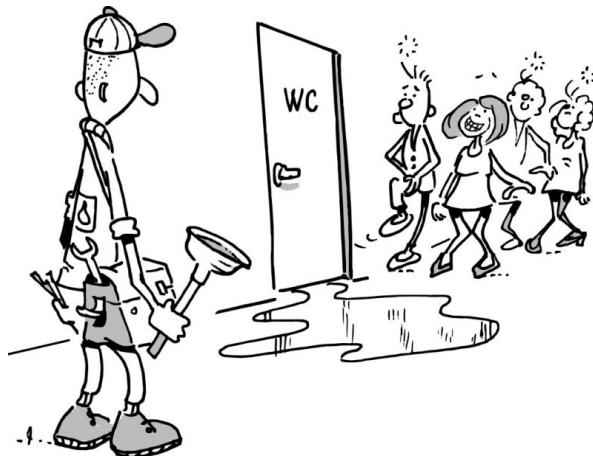


Fig. 3.4 Même la meilleure équipe peut avoir besoin d'un expert.

Bien démarrer

La question à se poser

Est-ce que la liste des membres de l'équipe et des parties prenantes y compris les experts est établie avec précision ?

De mauvais signes

Il n'y a pas d'expert identifié alors que l'équipe en a besoin.

Les parties prenantes ne sont pas sensibilisées à Scrum. Elles ne savent même pas que cela implique d'avoir ce rôle.

Par quoi démarrer

Une lecture pour tous

Un développeur est dans deux équipes en même temps.

Chaque développeur travaille sur son « module ».

Les membres de l'équipe reçoivent les compléments de formation nécessaires, pour s'habituer à travailler tous ensemble. Cela est fait pendant le sprint zéro. Les valeurs de l'équipe sont affichées sur le mur.

Un livre engagé et inspirant sur les gens de Scrum, qui reprend des essais publiés par Tobias Mayer sur ses blogs [Mayer, *People's Scrum*]. Une traduction en français est annoncée.

À retenir

Les gens sont ce qu'il y a de plus important dans Scrum.

Les gens qui sont dans l'équipe ont une place centrale. Scrum adopte l'idée d'organisation sans hiérarchie autoritaire : l'équipe est investie du pouvoir et de l'autorité pour faire ce qu'elle doit et s'organise par elle-même. C'est une des différences majeures avec les méthodes traditionnelles.

L'objectif d'une équipe Scrum est de produire des résultats pour ses parties prenantes. Celles-ci apportent leur feedback, apportant ainsi de la connaissance à l'équipe.

Les experts sont des parties prenantes qui aident l'équipe à réaliser le travail quand elle n'est pas pluridisciplinaire. Ponctuellement, c'est utile à l'équipe, mais cela ajoute des dépendances.

Références

☞ Tobias Mayer, *The People's Scrum*, 2013.

<http://agilelib.net/?100000>

☞ Véronique Messager, *Coacher une équipe agile*, 2012, Eyrolles.

☞ Dan Rawsthorne & Doug Shimp, *Exploring Scrum*, 2012.

<http://agilelib.net/?100002>

☞ Pablo Pernot, *La horde agile*, essai, 2014.

<http://agilelib.net/?1392302896.85>

Notes

- [1] Je les ai utilisées lors du mariage de ma fille comme support à mon discours.
- [2] Les Américains parlent d'équipe à deux pizzas <http://blog.jasoncrawford.org/two-pizza-teams> et on peut remarquer que le nombre de participants à une mêlée de rugby est dans cette fourchette.
- [3] Salle dédiée, une pratique du référentiel de l'Institut agile : <http://referentiel.institut-agile.fr/teamroom.html>.

Le rôle du Product Owner

Il y a quelques années, je travaillais chez un éditeur de logiciel.

Je faisais du marketing (tout le monde peut se tromper) pour un produit, aujourd’hui disparu, comprenant un éditeur graphique, un simulateur et un générateur de code. La cible principale était le secteur des télécoms.

J’avais été moi-même développeur et chef de projet dans le domaine des télécoms, donc je connaissais bien les utilisateurs potentiels du produit. Avec le rôle que j’avais, je les représentais en quelque sorte. J’essayais de faire prendre en compte leurs besoins dans le produit.

Le produit était développé par une équipe de la direction technique. Je ne la voyais pas très souvent, je ne les connaissais même pas tous. Je voyais un peu plus souvent le chef de projet. Lui et le directeur technique avaient aussi leurs idées, bien arrêtées, sur le produit.

Comme c’était une société dirigée par la technique, ils avaient souvent le dernier mot pour inclure ce qu’ils avaient décidé dans le produit. Les commerciaux allaient les voir et, si cela ne suffisait pas, s’adressaient directement au PDG pour ajouter leur point de vue.

Le résultat de ces apports différents à la définition du produit était un manque d’homogénéité. Le produit comportait des fonctions très puissantes. Par exemple, de la simulation exhaustive de modèles basés sur des machines à états. Il n’était pas simple à utiliser, et à mon avis, les fonctions proposées ne tenaient pas réellement compte des besoins des utilisateurs.

C’est pour éviter ce manque d’unité dans le contenu d’un produit que le **Product Owner** existe dans Scrum. Sa raison d’être est de faire en sorte que le travail réalisé apporte de la valeur aux utilisateurs. L’existence du Product Owner permet aussi d’éviter, comme dans mon histoire, la prédominance de la technique ou de personnes peu impliquées dans le développement.

À l’époque, si le rôle avait existé, j’aurais bien aimé être Product Owner. Le temps a passé et depuis que je pratique Scrum, c’est un rôle que j’ai exercé : en particulier, j’ai été, pendant de nombreuses années, le Product Owner du produit iceScrum.

C’est de ce rôle et de mes expériences dont il est question dans ce chapitre. Mais commençons par du vocabulaire.

J’ai d’abord connu Scrum par la lecture de livres et d’articles, tous en anglais. J’ai alors naturellement traduit Product Owner en « propriétaire de produit » quand je m’exprimais à propos de ce rôle.

En 2005, j'ai suivi une formation Scrum donnée par François^[1], un Québécois. Le support de cours était en français. Product Owner, dont on parlait assez peu d'ailleurs, y était traduit en « administrateur du produit ». Cet intitulé n'a jamais pris en France et je ne crois pas que nos cousins du Canada l'aient conservé.

En 2006, suite à la lecture d'un article [Marick, *Product Director*], j'ai adopté le terme **directeur de produit**. J'en ai parlé dans plusieurs billets de mon blog, pendant quelques mois.

L'article en français de Wikipedia sur Scrum a repris le terme. Ce qui fait que cette traduction de Product Owner a eu un certain succès. La version française de « *Scrum et XP depuis les tranchées* »[Kniberg, *Tranchées*] l'a reprise également.

Pourtant, j'ai arrêté d'utiliser directeur de produit pour revenir à l'anglais Product Owner (avec une prononciation à la française, voire à la toulousaine). Le mot « directeur » ne passait pas dans des organisations : celui qui tenait le rôle n'était pas un directeur au sens hiérarchique. Le Product Owner donne bien la direction, mais il n'a pas de responsabilité hiérarchique sur des personnes.

L'appellation **Product Owner**, PO en abrégé, est maintenant complètement adoptée par la communauté française de Scrum.

Voici ma définition du rôle :

Le Product Owner est la personne de l'équipe, et la seule, qui soit imputable du résultat du produit auprès des parties prenantes.

Il convient de considérer le mot « produit » dans une acceptation large, comme le résultat de ce qui est fait par l'équipe. En fait il importe peu que l'équipe développe un produit ou un service ou une application : le Product Owner est responsable du résultat.

4.1. Responsabilités du Product Owner

Il agit à la fois sur la stratégie et sur la tactique.

Il prend des décisions de niveau stratégique qui étaient anciennement du ressort du chef de projet ou des comités de pilotage, comme le choix de la date de livraison du produit ou d'une fonctionnalité majeure à développer en priorité.

Le Product Owner prend aussi de nombreuses décisions de niveau tactique qui étaient, faute de présence « métier », prises par une équipe de développement, comme par exemple définir un emplacement ou un libellé sur une page web. En effet, pour avancer, les développeurs étaient contraints de faire des choix qui n'étaient pas, en principe, de leur responsabilité.

Un PO est souvent amené à prendre des décisions tactiques, c'est pourquoi sa présence dans l'équipe est indispensable.

Le Product Owner ne décide pas tout seul ; beaucoup de travaux qu'il mène se font en équipe et ses décisions sont prises, chaque fois que c'est important, en accord avec elle.

Bien que le rôle de Product Owner varie beaucoup avec le contexte de l'organisation, il est possible d'identifier ses activités majeures.

4.1.1. Partager la vision

On ne demande pas à un Product Owner d'être visionnaire comme Steve Jobs ; il doit cependant avoir une bonne vision du produit. Celle-ci se construit au début du développement d'un nouveau produit et se consolide ensuite. Elle consiste typiquement à définir :

- la raison pour laquelle on crée le produit,
- l'objectif de la prochaine release,
- les impacts attendus sur les acteurs,
- la liste des fonctionnalités essentielles qui y contribuent.

Tous les membres de l'équipe et toutes les parties prenantes du projet devraient partager la même vision, et c'est au PO de s'en assurer. Le mieux est d'ailleurs de bâtrir la vision avec eux, en les faisant participer à des ateliers et à des jeux (voir le chapitre 14 *Découvrir le produit*).

4.1.2. Affiner le produit

Le Product Owner définit le contenu du produit. Pour cela, il identifie les fonctionnalités requises, collectées auprès des parties prenantes et mises dans une liste, appelée **backlog**.



Fig. 4.1 Le PO ordonne le backlog

Concrètement, le Product Owner s'occupe du backlog et passe une bonne partie de son temps à l'affiner, en anticipation sur les sprints suivants.

4.1.3. Planifier

C'est le Product Owner qui définit l'ordre dans lequel les parties du produit seront développées. Il alimente l'équipe avec les fonctionnalités à développer, en s'efforçant de maximiser la valeur.

C'est aussi lui qui définit les jalons du cycle de vie du produit (voir le chapitre 16 *Planifier la release*). Il donne l'objectif d'une release et prend les décisions sur son contenu et sa date de mise à disposition auprès des utilisateurs.

En résumé, s'il n'a pas d'autorité formelle sur des personnes, le Product Owner a une grande influence sur le produit réalisé.

4.2. Compétences souhaitées

Dans l'idéal, la personne qui joue ce rôle devrait posséder des compétences variées.

4.2.1. Bonne connaissance du métier

Ce qu'on appelle le métier (*business*), et qu'on retrouve en français dans l'expression « cœur de métier », se rapporte à un domaine de connaissances en relation avec les utilisateurs du produit.

Une bonne connaissance du métier est fondamentale pour le Product Owner, puisqu'il est le représentant dans l'équipe de toutes les personnes qui utilisent ou vont utiliser le produit.

Il peut l'avoir acquise parce qu'il est l'un des utilisateurs, et c'est souvent le cas dans les entreprises qui développent des produits à usage interne. Dans celles produisant pour des clients externes, il vient souvent des équipes marketing ou produit.

On ne demande pas à un Product Owner de tout connaître du domaine fonctionnel : sur des grands projets, cela serait une gageure ou demanderait beaucoup de temps. Il s'appuiera, quand cela s'avérera nécessaire, sur les bonnes personnes. Si elles sont nombreuses, une d'elles peut jouer le rôle d'interlocuteur privilégié avec lui.

4.2.2. Maîtrise des techniques de définition de produit

Le Product Owner définit ce que fait le produit. Cela requiert la maîtrise des techniques de collecte des besoins et de leur transformation en éléments du backlog.

Scrum ne prescrit pas de technique particulière pour identifier les éléments du backlog, cependant on constate sur le terrain que la pratique la plus fréquemment associée à Scrum est la « story », que le PO se doit de bien connaître.

4.2.3. Autorité pour prendre des décisions rapidement

Choisir entre plusieurs alternatives sur des sujets d'importance variée, cela arrive quotidiennement sur un projet : pour des raisons d'efficacité, le Product Owner doit

pouvoir le faire seul, sans en référer à une hiérarchie ou à une autorité supérieure. Pour cela, il est essentiel qu'il ait la confiance de l'ensemble des parties prenantes et des développeurs.

Si les avis de ces intervenants sont contradictoires, le Product Owner possède l'autorité, par délégation, pour décider. Décider, c'est choisir ; c'est aussi dire non à des demandes des parties prenantes.

Le PO entraîne l'équipe en faisant en sorte que tout le monde adhère pleinement aux choix sur le contenu du produit.

4.2.4. Capacité à détailler au bon moment

Avec une approche agile, tout le travail de définition n'est pas fait dès le début du développement. Le Product Owner doit donc savoir, en fonction de l'avancement, à quel moment approfondir des éléments du backlog.

Il sait anticiper. La notion de priorité entre les éléments du backlog va l'y aider : ce qui est le plus prioritaire doit être prêt en premier.

Le Product Owner affine, au bon moment, le contenu du backlog.

4.2.5. Esprit ouvert au changement

L'outil principal du Product Owner, le backlog, vit et évolue pendant toute la vie du projet. Cela constitue une différence fondamentale avec la pratique souvent ancrée dans la culture des organisations qui consiste à essayer de figer les spécifications au début.

Un PO sollicite et prend en compte le feedback, qui remonte suite aux livraisons régulières des versions du produit aux parties prenantes. Ajuster l'orientation du produit avec ce feedback est la preuve de son ouverture d'esprit.

Il ne faut pas comprendre la capacité au changement comme le droit de changer d'avis à tout moment. Un Product Owner garde le cap : sa vision ne varie pas fondamentalement pendant une release. Bien qu'ouvert au changement, il n'est pas une girouette.

4.2.6. Aptitude à la négociation

Le Product Owner décide des priorités en fonction de la valeur ajoutée ; cependant ce n'est pas le seul paramètre à prendre en compte. En particulier au début d'une release, il existe des risques techniques et, plus tard, de la dette technique. Un dialogue avec les développeurs est nécessaire pour pondérer les critères de priorité.

Lors des séances d'affinage qu'il effectue en compagnie de l'équipe, le Product Owner fait preuve de pédagogie pour expliquer ce qu'il propose d'ajouter au produit et prend le temps d'écouter les réactions.

Pendant le sprint, il est indispensable qu'il rencontre souvent le reste de l'équipe. Il vient à la mêlée et c'est une bonne habitude qu'il reste discuter avec les développeurs à la suite de cette réunion. Ces conversations permettent de confirmer que la fonction à développer a été bien comprise par tous.

Il lui arrivera de constituer un binôme avec un développeur, pour que celui-ci lui montre, sur son environnement de développement, le fonctionnement de la story sur laquelle il travaille et lui fasse part d'une nouvelle idée pour l'améliorer.



Fig. 4.2 Le PO va discuter avec un développeur.

Le Product Owner négocie fréquemment avec l'équipe ; il prend le temps d'expliquer ses prises de position.

4.3. Choisir le Product Owner d'une équipe

Au moment où l'équipe Scrum se constitue, il faut trouver la bonne personne pour tenir le rôle. Les organisations étant extrêmement diverses, les possibilités de choix sont très variables.

On peut se baser sur les compétences souhaitées du Product Owner présentées plus haut. Seulement, c'est un défi de trouver quelqu'un qui possède toutes ces qualités, et même si on le trouvait, il faudrait encore qu'il soit très disponible.

4.3.1. Une personne disponible

En effet, la disponibilité pour le rôle est une condition *sine qua non* pour qu'une personne devienne Product Owner. Par opposition à ce qui se passe pour un projet classique, où l'intervention du client (ou de son représentant) est importante au début et à la fin mais pas entre les deux, l'implication du PO est continue sur tous les sprints.

Disponibilité continue

On estime que le travail nécessite une participation d'au moins trois heures par jour à disposition d'une équipe de cinq personnes.

On peut même considérer que, plus la fin de la release se rapproche, plus le temps que le Product Owner devrait consacrer au projet augmente : en effet, les priorités deviennent plus difficiles à décider, les tests à passer se font

plus nombreux...

Participation aux événements Scrum

Le Product Owner fait partie de l'équipe. Il participe aux réunions du sprint.

Le constat fait sur le terrain est que cette règle, pourtant essentielle, n'est pas toujours respectée. Parfois, au moment de choisir le Product Owner, il arrive qu'on me demande quelle est son implication minimale : comme la personne susceptible de tenir le rôle a d'autres activités (par exemple, il est également utilisateur de l'ancienne version du produit), on cherche, à tort, à limiter le temps qu'il passera sur le projet, en particulier dans les réunions avec l'équipe.

Pour des sprints de deux semaines, voilà une idée de sa participation :

- réunion de planification de sprint : environ deux heures ;
- mêlées quotidiennes, deux fois par semaine (c'est un minimum) : 60 minutes pour quatre séances ;
- affinage : environ trois heures ;
- revue de sprint et rétrospective : une heure chacune.

Cela représente huit heures, soit environ 10 % de son temps.

Implication au jour le jour

En plus de participer à ces réunions, le Product Owner se doit aussi de :

- affiner seul le backlog, ajuster les priorités ;
- répondre aux questions sur le produit ;
- définir les conditions d'acceptation ;
- vérifier la terminaison d'une story.

4.3.2. Une seule personne

Parler d'une seule voix

Le PO est le seul décideur pour les questions qui relèvent de ce que doit faire le produit.

Une équipe Scrum avait demandé à me rencontrer pour me poser des questions sur le produit, après une réunion qu'ils avaient eue avec leur Product Owner, qui les avait envoyés vers moi. Il se trouve que je connaissais bien le domaine. J'ai donc rencontré l'équipe et répondu à leurs questions. Les réponses que j'ai données étaient orientées selon mes idées sur le produit.

Seulement leur Product Owner leur avait demandé de me voir pour une seule question, bien précise. Je ne l'ai su qu'après, l'équipe ne me l'a pas dit et m'a posé plein d'autres questions.

Résultat : j'ai orienté l'équipe dans une voie qui n'était pas celle de son PO et celui-ci a eu ensuite beaucoup de mal à s'affirmer et à mettre en avant sa vision.

Il faut parler d'une seule voix à une équipe, et cette voix, c'est celle du Product Owner.

Déléguer, sauf ce qui est essentiel

Comme il est difficile de trouver toutes les qualités attendues dans une seule personne, peut-on les partager entre plusieurs ? Si le Product Owner n'est pas suffisamment disponible pour assumer ses responsabilités, est-il alors envisageable d'avoir dans l'équipe un PO délégué ?

Exemple : dans le cas d'une société de services qui réalise un produit pour un client, le PO est normalement dans l'organisation du client. Si l'équipe considère qu'il n'est pas suffisamment présent, elle sera tentée de désigner un PO délégué en son sein.

Cela peut fonctionner un temps, mais le risque dans ce cas est de soumettre l'équipe à deux sons de cloche et de la perturber quand le vrai PO s'exprimera après son délégué.

Un PO proxy n'est qu'un pis-aller qu'il est préférable d'éviter.

Si le PO officiel ne s'implique pas assez, mieux vaut avoir une approche radicale : le considérer comme une partie prenante et désigner le délégué comme le véritable PO, au moins jusqu'à la fin de la release courante.

Ce qui définit le rôle, c'est l'imputabilité devant les parties prenantes des résultats de l'équipe, avec un moyen essentiel pour l'exercer : définir les priorités.

Le PO est une seule personne, pas un comité

Dans de nombreuses organisations, il semble difficile de trouver une personne suffisamment disponible pour être Product Owner. Apparemment il serait plus facile d'en trouver plusieurs à temps partiel pour se partager le rôle. Il y a aussi des cas où la connaissance est disséminée entre plusieurs personnes.

Plusieurs Product Owners, c'est la porte ouverte aux conflits à cause d'intérêts contradictoires, aux décisions ralenties, aux difficultés à prioriser. C'est pour éviter cela que Scrum recommande une seule personne pour ce rôle. Comme le dit Ken Schwaber :

« *Le Product Owner est une personne, pas un comité.* »

Cependant, il ne s'agit pas de s'en tenir à une position dogmatique : on peut avoir deux PO pour une équipe, s'ils jouent ce rôle dans l'esprit.

D'ailleurs, il n'est pas possible de garder un seul PO dans les grands projets. Ce sujet fait l'objet du chapitre 21 *Développer un produit avec plusieurs équipes*.

4.3.3. Où le trouver dans l'organisation actuelle ?

Le Product Owner est plutôt à chercher dans un service qui est en contact avec les utilisateurs ou qui les représente, donc généralement une entité autre que celle des membres de l'équipe.

Par exemple, chez un éditeur de logiciel, il est probable qu'il soit issu du service marketing ou produit. Venant d'une entité où l'idée de produit est déjà très forte, il y a de bonnes chances que cette personne possède quelques-unes des compétences requises pour être un bon PO.

C'est dans ce cadre que j'ai rencontré les meilleurs Product Owners, avec une bonne culture produit.

Dans les grandes entreprises qui ont des utilisateurs internes, il paraît logique qu'un PO soit choisi dans le service où sont les utilisateurs et leurs représentants. Quelqu'un qui a été analyste métier (*Business Analyst*), en assistance à ce qu'on appelle la maîtrise d'ouvrage (AMOA), ou encore chef de projet utilisateurs est un bon candidat.

Dans ce type d'organisation, j'ai cependant rencontré des Product Owners qui avaient plus de difficultés à bien jouer leur rôle, en particulier par manque de disponibilité.

Dans les organisations qui ont traditionnellement un chef pour chacun de leurs projets, la logique voudrait que le chef de projet devienne PO lors du passage à Scrum. En effet, PO est bien un rôle de leader (mais pas de chef !), portant la vision et décisionnaire sur le produit.

4.3.4. Une personne motivée pour le rôle

Le Product Owner fraîchement nommé a parfois une culture de Scrum et de l'agilité superficielle. Il pourra recevoir des compléments de formation plus tard, mais ce qui importe lors du choix, c'est son adhésion aux valeurs et principes véhiculés par le mouvement agile.

On ne peut pas être un bon PO si on n'est pas motivé pour le rôle. Heureusement la motivation vient vite, car c'est un rôle passionnant.

4.4. Une journée typique de PO

Céline tient le rôle de Product Owner de l'équipe Peetic.

4.4.1. Collaborer avec l'équipe

En tant que PO, Céline n'est pas simplement le représentant des clients et des utilisateurs dans l'équipe. Elle est aussi membre de l'équipe et participe activement aux travaux pendant un sprint.

La présence physique du PO avec l'équipe a un impact sur son rôle :

- La situation idéale est un PO installé avec l'équipe, dans le même espace de travail. Cependant, il suffit qu'il soit assez proche pour que l'équipe puisse aller le voir et lui demander des précisions, et dans l'autre sens, pour qu'il puisse rencontrer facilement l'équipe.
- Si le PO est à distance et ne voit pas physiquement l'équipe, la mise en œuvre du rôle sera plus délicate, mais il existe des solutions pour pallier son manque de présence

physique.

En collaborant avec l'équipe, Céline apprend à écouter ce qu'ont à dire les développeurs. Elle comprend mieux leurs préoccupations de délai et de qualité. Ainsi, elle sera moins enclue à leur imposer une pression négative en les poussant à livrer toujours plus de fonctionnalités.

4.4.2. S'impliquer dans l'acceptation du fini

Comme il est moteur pour établir ce que doit faire le produit, il est logique que le PO fournisse aussi les conditions qui permettront de s'assurer que ce qu'il demande est bien terminé.

Lors des séances d'affinage, Céline a formulé à l'équipe le comportement qu'elle attend d'une fonctionnalité. Ensemble, ils ont identifié une condition d'acceptation. C'est en vérifiant que cette condition est satisfaite qu'elle s'assure de sa bonne fin.



Fig. 4.3 On ne laisse pas le PO tout seul avec son backlog

4.4.3. Utiliser le produit

Céline aime son produit. Elle l'utilise tous les jours, ce qui l'incite à en faire un bon produit. Cela lui permet de discerner la valeur ajoutée par les fonctions dans le backlog. Bien connaître son produit lui donnera une position respectée par tous et facilitera ses prises de décision.

C'est aussi dans sa mission de faire des démonstrations à l'extérieur et de présenter le produit aux utilisateurs. Elle en profite pour valider les hypothèses sur les nouveautés qu'elle envisage d'inclure dans le produit.

4.4.4. Impliquer les parties prenantes

Céline est imputable du résultat auprès des parties prenantes : pour bien exercer son rôle, elle doit communiquer avec elles.

Cela concerne non seulement les utilisateurs du produit final, mais aussi les clients comme les vétérinaires, et Gilles et Mona, bien sûr. Elle les invite à la revue de chaque fin de sprint. Elle doit les inciter fermement, car la revue est un moment d'inspection collectif de l'état du produit.

4.5. Le PO sur le terrain

4.5.1. Se former au rôle de Product Owner

Devenir un bon Product Owner nécessite des compétences particulières, qu'une formation spécifique au rôle peut aider à acquérir. Ceux qui ont appris Scrum sur le tas, notamment, peuvent avoir besoin d'être recadrés sur les fondamentaux.

Une technique très utile au PO pour décrire les éléments du backlog est celle des *user stories*. Elle facilite la gestion de projet et le dialogue avec les utilisateurs. En complément, l'acquisition des pratiques de définition de produit lui permettra d'appréhender une approche utile pour le développement d'un nouveau produit. Une bonne formation doit inclure l'apprentissage de ces techniques, de préférence avec des ateliers de travail en groupe.

Le PO qui recherche une formation aura intérêt à vérifier qu'elle est animée par un formateur qui a de l'expérience dans ce rôle et dans l'accompagnement sur le terrain.

Une fois formé, le PO comprendra mieux pourquoi son rôle demande une grande disponibilité et le pouvoir de décider. Il aura plus d'arguments pour les obtenir.

Pour apprendre concrètement le rôle, en particulier dans les organisations où le clivage entre utilisateurs et informaticiens est fort (de type MOA/MOE), il est plus efficace d'être accompagné. Un coach externe (ou le ScrumMaster) apporte sa connaissance et son expérience du rôle, ce qui permet de progresser rapidement grâce à un travail en binôme.

4.5.2. S'intéresser aux nouvelles approches

Impact mapping, story mapping, innovation games ... sont des outils qui permettent au PO de dépasser Scrum. Nous en parlerons plus loin.

Pour partager sur son métier, il trouvera de nombreuses sessions dans les conférences qui portent sur le rôle et sur les nouvelles approches.

En interne, il pourra rejoindre la communauté de pratiques des PO, ou la créer si elle n'existe pas.

4.5.3. Planifier à moyen terme

Une des pratiques de Scrum les plus mal utilisées, d'après mon expérience, est la production d'un *plan de release*. C'est pourtant un moyen de donner une direction à l'équipe et de la communiquer aux parties prenantes. Le Product Owner est moteur dans cette quête d'avoir un horizon qui va au-delà du sprint courant.

Bien commencer

La question à se poser

Est-ce que le PO est bien en capacité à prendre des décisions ?

De mauvais signes

Le PO ne vient pas à la mêlée quotidienne.

C'est lui seul qui écrit les stories.

Par quoi démarrer

Collaborer avec l'équipe. Cela est fait dès le sprint zéro.

Une lecture pour tous

Confessions d'un « serial Product Owner » [Forss, Confessions], un essai qui date un peu, mais qui présente bien l'essentiel du rôle. Il est traduit en français.

À retenir

Dans une équipe Scrum qui développe un produit, le Product Owner est le responsable du résultat auprès des parties prenantes. Il apporte sa vision à l'équipe et définit les priorités de façon à obtenir un produit apportant le maximum de valeur.

L'implication d'un bon Product Owner est capitale pour assurer le succès du projet. En définissant sa vision sur le produit, il donne l'impulsion à l'équipe. En faisant la promotion du résultat de chaque sprint auprès des utilisateurs, il fournit à l'équipe une reconnaissance qui la motive. En collaborant avec l'équipe, il fait converger les énergies pour maximiser la valeur ajoutée au produit.

Références :

☞ Anna Forss, *Confessions d'un serial Product Owner*, Essai, Web.

<http://wiki.ayeba.fr/Confessions+d%27un+serial+product+owner>

☞ Henrik Kniberg, *Scrum et XP depuis les tranchées*, traduit en français, 2009.

<http://henrik-kniberg.developpez.com/livre/scrum-xp/>

☞ Brian Marick, *How to be a Product Director*, Essai, 2006.

<http://www.example.com/writing/product-director.pdf>

Notes

[1] François Beauregard, préfacier des précédentes éditions de cet ouvrage.

Le rôle du ScrumMaster

Lorsqu'on évoque un projet développé par un groupe, une pensée très répandue est de considérer qu'une personne identifiée doit être responsable de l'équipe. Traditionnellement, ce rôle est appelé **chef de projet**. En France, ce rôle est solidement ancré dans la culture du développement. En voici deux exemples :

- Beaucoup d'étudiants en informatique, passant un entretien pour rentrer dans une école, mettent un point d'honneur à dire que leur objectif est de devenir chef de projet dès leur entrée dans la vie professionnelle. Probablement parce que des enseignants croyant bien faire leur ont inculqué cette notion de l'ambition.
- Récemment, au cours d'une présentation de Scrum dans une grande entreprise publique, tous les participants se sont présentés, lors du tour de table, comme chefs de projet. Souvent dans les entreprises qui font appel à la délégation de personnel, il ne reste que des chefs de projet dans l'organisation, surtout responsables des résultats.

Invité à un ScrumDay, Dominique Dupagne, médecin, auteur [Dupagne, Rameur] et chroniqueur dans la « Tête au carré » sur France Inter, avait mis en exergue cette tendance des organisations à se doter de strates de chefs au risque de ne conserver que peu de monde produisant réellement de la valeur.

Pas de chef de projet dans Scrum ! Le rôle est éliminé.

Le travail et les responsabilités d'un chef de projet ne disparaissent pas pour autant dans les projets Scrum. Une partie est dévolue au Product Owner, qui est responsable des résultats, une autre est laissée à l'équipe. L'auto-organisation signifie que les membres de l'équipe s'organisent eux-mêmes et n'ont pas besoin d'un chef qui leur assigne le travail à faire. **ScrumMaster** n'est donc pas un nouveau nom pour chef de projet.

On utilise souvent des analogies pour expliquer le rôle de ScrumMaster : berger, capitaine, bouledogue, etc.

Dans ma version rugbystique, c'est le demi de mêlée. La mêlée se réfère aux membres du pack dans le rugby à quinze. Le demi de mêlée fait avancer son pack lors d'un maul, le guide dans la progression, demande le ballon au bon moment.

Certaines interlocutrices m'ont fait remarquer que l'image virile véhiculée par le rugby n'était pas de nature à attirer des développeuses. Peut-être, mais c'est dommage, car le rôle est débarrassé des oripeaux du chef traditionnel, le plus souvent associé à l'image du mâle dominant.

5.1. Responsabilités du ScrumMaster

Voici ma définition du rôle :

Le ScrumMaster (SM) est une personne dans l'équipe Scrum qui se met à son **service**, pour faciliter la réalisation des travaux demandés par le Product Owner, en **appliquant Scrum** au mieux, compte tenu du **contexte** de l'organisation.

5.1.1. Servir l'équipe

Une des missions du SM est de motiver l'équipe pour qu'elle s'auto-organise. Il fait tout pour que l'équipe progresse.

Il pousse l'équipe à devenir pluridisciplinaire, en renforçant ses capacités en ingénierie, pour ne plus dépendre d'experts extérieurs.

S'il réussit, l'équipe aura moins besoin de lui, c'est le paradoxe du SM.

Alors que l'implication d'un Product Owner est toujours constante, celle d'un ScrumMaster a tendance à diminuer dans le temps.

5.1.2. Éliminer les obstacles

Il se produit toujours des événements imprévus pendant un développement. Certains sont susceptibles de ralentir ou de bloquer le travail de l'équipe. Dans le jargon Scrum, ils sont appelés des **obstacles** (*impediments*), et peuvent être de nature et d'importance très variables.

Un obstacle est un fait concret touchant une ou plusieurs personnes et qui empêche l'équipe d'avancer à son rythme.

Exemples : dans l'équipe Peetic, un développeur s'est cassé le bras au ski, le serveur Git est tombé en panne, le composant attendu pour le paiement en ligne n'est pas prêt, le Product Owner ne répond pas, etc.

C'est au ScrumMaster de pousser l'équipe à mettre en évidence les obstacles, et c'est aussi à lui de s'assurer de leur élimination.

Il fait en sorte d'éviter qu'ils ralentissent durablement l'équipe. Il s'appuie sur des compétences internes à l'équipe ou va en chercher à l'extérieur si c'est nécessaire pour résoudre un problème.

5.1.3. Appliquer Scrum

Le SM aide à progresser avec Scrum et à l'appliquer, dans le respect des valeurs d'équipe. Il enseigne les pratiques jusqu'à ce que l'équipe les mette en œuvre naturellement.

L'originalité de Scrum, parmi les pratiques de management, vient du fait que les responsabilités sont partagées : le PO prévoit et anticipe, tandis que le SM accompagne l'équipe qui réalise ce que demande le PO.

La réussite de Scrum repose sur la tension de la demande entre le PO et l'équipe, tension **contrôlée** de façon positive par le ScrumMaster.

5.1.4. Pratiquer l'art du possible

Le SM a pour mission de faire appliquer Scrum, mais une posture trop radicale face au management peut conduire au rejet de Scrum. Il doit tenir compte du contexte de l'organisation.

En particulier, le SM protège l'équipe des perturbations, mais il doit savoir jusqu'où il est possible d'aller, face à une organisation qui n'arrive pas à changer ses habitudes rapidement.

5.2. Compétences souhaitées

5.2.1. Bonne connaissance de Scrum

Le ScrumMaster est la personne qui est supposée maîtriser Scrum, plus que les autres. Au-delà de la simple connaissance théorique de Scrum, il est préférable qu'il ait déjà une expérience de sa mise en œuvre, pour éviter d'appliquer des règles sans discernement, car il est toujours nécessaire de s'adapter au contexte.

Sa connaissance ne doit pas s'arrêter à son rôle, mais englober l'ensemble du cadre Scrum. En particulier, il est le garant des valeurs et les promeut auprès de l'équipe.

5.2.2. Aptitude à comprendre le fonctionnel et la technique

Formellement, il n'est pas nécessaire pour un ScrumMaster de bien connaître le domaine de l'application à développer. Toutefois, une expérience dans le « métier » facilitera la communication avec le Product Owner et permettra de mieux impliquer l'équipe dans la recherche de la valeur pour le produit.

On ne demande pas non plus à un ScrumMaster d'être un « cador » en technique. Il s'appuie sur des experts pour les aspects techniques pointus. Cependant, des connaissances dans les technologies utilisées permettent de mieux appréhender les problèmes rencontrés par son équipe. Cela facilite la communication, en particulier avec les développeurs, et rend plus aisée l'identification des obstacles qu'ils rencontrent.

5.2.3. Facilité à communiquer

Des talents de communication sont nécessaires, car le ScrumMaster est amené à discuter fréquemment avec l'équipe ainsi qu'avec le management.

Ces discussions ont lieu dans différents contextes, ce qui nécessite de sa part d'adapter le style de communication :

- il sait obtenir la confiance quand il est en face à face avec un membre de l'équipe ;
- il fait en sorte que les événements du sprint, en présence de nombreuses personnes, se déroulent efficacement ;

- il est tenace et ferme dans ses demandes au management, sans pour autant être intransigeant.

5.2.4. Capacité à guider

Il influence l'équipe : c'est un meneur, un guide qui sait créer les conditions pour que l'équipe soit motivée, pour qu'elle arrive au résultat. Mais il doit arriver à ses fins par la conviction, sans imposer ses décisions : un ScrumMaster ne dispose pas d'autorité hiérarchique sur les membres de l'équipe.

Pendant le sprint zéro, le SM peut être impliqué dans la constitution de l'équipe. Il est le garant des valeurs et fait en sorte que l'équipe en soit bien imprégnée. C'est également à lui de s'assurer que la logistique, en particulier les bureaux et leur agencement, est adaptée aux pratiques de travail en équipe.

Pendant un sprint, il accompagne l'équipe vers le respect de l'engagement, en la focalisant sur l'objectif du sprint défini en commun.

5.2.5. Talent de médiateur

Son travail le plus important, en durée, est d'éliminer les obstacles. Parmi ceux-ci, un certain nombre est dû à des conflits entre personnes.

Lors d'un différend entre des membres de l'équipe, il joue le rôle de médiateur pour aider les gens concernés à trouver une solution acceptable. Il pousse au consensus.



Fig. 5.1 Un SM qui fait le médiateur

En cas de désaccord persistant, il propose une mesure plus radicale, comme changer une personne d'équipe. En cas de conflit avec le Product Owner, il fera attention de ne pas (re)créer une opposition entre les développeurs et les utilisateurs : le Product Owner est dans l'équipe pour éviter cette fracture.

J'ai connu un ScrumMaster qui avait mal compris son rôle. Sous prétexte de considérations techniques, il s'opposait au Product Owner, essayant d'empêcher une mise en production. S'il est normal qu'il existe une tension entre les deux rôles, ce n'est pas le ScrumMaster qui est responsable de la vie du produit. Il se limite à exposer le point de vue de l'équipe.

5.2.6. Ténacité

Le ScrumMaster fait son possible pour éviter que des obstacles aient un impact sur la progression de l'équipe. Parfois, ils ne peuvent être éliminés que par l'intervention de personnes faisant partie d'autres équipes ou par le management. Ces personnes sont souvent difficiles à rencontrer et encore plus à convaincre d'agir rapidement. Un ScrumMaster n'abandonne pas à la première adversité. Il se montre opiniâtre, il poursuit sa quête jusqu'à l'élimination de ce qui freine l'équipe.

5.2.7. Inclination à la transparence

Scrum pousse à la transparence. Le ScrumMaster en est le garant.

À la différence d'un chef de projet, il est davantage sur l'accompagnement de l'équipe que sur le suivi individuel : les mesures faites avec Scrum sont collectives.

Les chefs de projet traditionnels ont tendance à faire beaucoup de *reporting*. Avec Scrum, la façon de produire des indicateurs est différente et cela est fait rapidement : pas besoin de passer beaucoup de temps à faire des consolidations.

En tant que garant de la transparence, le SM fait le nécessaire pour que les indicateurs soient publiés et compris par les parties prenantes. Il n'a pas de responsabilité particulière pour produire ce *reporting*.

En revanche, il est responsable de remonter les obstacles majeurs. D'ailleurs, un apport fondamental de Scrum est de révéler les dysfonctionnements au plus tôt. Le devoir du ScrumMaster est de les mettre en évidence, pour permettre à l'équipe de s'adapter à la situation.

5.2.8. Goût à être au service de l'équipe

Le ScrumMaster n'est pas un chef : il ne commande pas, il n'impose pas, il ne constraint pas. Il est au service de l'équipe, il lui offre son support.

Son humilité, qualité essentielle, consiste à ne pas se mettre en avant :

- si le sprint est un succès, ce n'est pas lui qui a réussi, c'est l'équipe ;
- si le projet a des difficultés, ce n'est pas la faute des autres membres de l'équipe.

5.3. Choisir le ScrumMaster d'une équipe

5.3.1. Une personne adaptée au niveau de l'équipe

La façon dont le rôle est joué dépend du degré de maturité de l'équipe.

Comme nous l'avons vu, un groupe qui se forme passe par des étapes successives ; à chaque niveau atteint correspond une application adaptée du rôle de ScrumMaster :

- d'abord, il apprend Scrum à l'équipe,
- ensuite, il guide l'équipe dans l'application de Scrum,
- puis il apporte des idées pour que les développeurs prennent des initiatives,
- et enfin, il les entraîne à faire émerger l'intelligence collective.

En résumant, on peut dire qu'au début, on prend une personne qui connaît bien Scrum et qu'après, on se tourne plutôt vers une posture de coach.

5.3.2. Quelqu'un de disponible

Les activités demandées au SM, en particulier l'élimination des obstacles, demandent du temps.

Pour une équipe Scrum typique qui démarre, la personne qui devient ScrumMaster joue ce rôle à plein temps.

Il fait partie de l'équipe : il s'engage avec les autres. Il doit régulièrement rencontrer – physiquement – les membres de l'équipe, il ne reste pas dans son bureau.

Dans de petites équipes, il peut aussi participer aux travaux de développement. Il prend alors des tâches du sprint comme les autres membres, mais cela doit rester limité : le rôle de ScrumMaster prend du temps et il est prioritaire sur ses autres tâches.

En revanche, il faut absolument éviter qu'une personne soit en même temps ScrumMaster et Product Owner de l'équipe et, au moins dans un premier temps, le ScrumMaster de plusieurs équipes.

5.3.3. Quelqu'un qui incarne le changement

Le terme ScrumMaster est sujet à caution, dans sa partie *Master*. Le langage influence le comportement : même si l'appellation ScrumMaster est nouvelle, le terme *master* n'aide pas toujours les organisations à changer de paradigme.

Dans certaines organisations à culture hiérarchique, le rôle de SM, *maître de Scrum* peut être perçu comme un rôle de responsable, dirigeant des personnes.

Scrum représente un changement radical avec ce rôle nouveau. Dans le cas d'organisation à culture hiérarchique forte, cela impacte les fondements de la gouvernance.

C'est pourquoi la personne devenant ScrumMaster doit avoir bien compris l'essence du rôle pour être l'incarnation du changement qu'il représente.

Pour certaines équipes, c'est un développeur expérimenté qui devient le SM. Mais dans la majorité des cas, c'est un ancien chef de projet qui a pris le rôle. Par exemple, dans les grandes organisations, le rôle de SM est pris naturellement par un chef de projet informatique.

On peut demander qui est volontaire pour jouer le rôle de SM. On peut aussi l'élire. La sociocratie nous apporte une nouvelle possibilité pour choisir le SM dans une équipe : l'élection sans candidat. On lira à ce sujet l'ouvrage collectif *Rupture Douce* [Sarrazin et coll.].

5.3.4. ScrumMaster, un état d'esprit

Certes, on peut se former à devenir ScrumMaster, cependant, la personne qui prend le rôle doit avoir un état d'esprit approprié.

Quelques traits de caractère permettent de le déceler :

- la capacité à percevoir les émotions dans l'équipe,
- la curiosité et l'envie d'apprendre,
- l'inclination à penser que les gens font de leur mieux dans leur travail,
- l'envie de changer les choses même si c'est difficile,
- l'orientation vers le collectif,
- le goût de la prise de risques.

Il m'est arrivé de rencontrer ces ScrumMasters « naturels ». Ceux dont on se dit, comme pour Obélix : ils sont tombés dedans quand ils étaient petits (dans le village gaulois, le bon profil de SM, c'est plutôt Astérix qui le possède).

Le ScrumMaster pousse l'équipe à mettre Scrum en application. Il organise et anime les événements du sprint. Il fait en sorte que ces réunions aient lieu et qu'elles soient efficaces. Il y joue un rôle de facilitateur, littéralement « celui qui facilite les choses ».



Fig. 5.2 Le SM, un facilitateur

5.4. Une journée typique de SM

Les pratiques évoquées dans cette journée seront détaillées dans les chapitres suivants.

Nicolas est le ScrumMaster de l'équipe Peetic. Il a été élu sans être candidat, mais il a accepté avec plaisir.

C'est le troisième sprint de la release Canigou (l'équipe nomme ses releases avec les sommets des Pyrénées).

Le matin, après avoir répondu à ses mails, Nicolas accueille les développeurs près de la machine à café. On discute du film de la veille puis, à 9 h 30, c'est la mêlée quotidienne, devant le tableau du sprint. Il s'assure que l'amélioration décidée lors de la rétrospective, faire en sorte que la mêlée ne dure pas plus d'un quart d'heure, soit réussie.

Tout de suite après la mêlée, il provoque une réunion avec Julien et « l'ingé système ». Il s'agit d'éliminer l'obstacle lié au serveur de « staging » qui ne fonctionne pas encore et empêche de déployer facilement à chaque sprint.

Une fois la solution trouvée, Nicolas met à jour le tableau des obstacles. Ouf, il n'y en a plus que trois à régler. En passant, il regarde si les tâches ont bien été mises à jour après la mêlée. C'est bon.

En début d'après-midi, comme tous les mercredis, ce sera la réunion d'affinage du backlog. Il a une conversation brève avec Céline le PO, afin de s'assurer qu'il y aura de quoi alimenter l'équipe pour le prochain sprint, pour éviter les à-coups dans le rythme.

À midi, il part courir au bord du canal.

Le temps de prendre la douche et la pâtée, c'est l'heure de la réunion d'affinage. On y a invité Laurent, l'expert en cartographie, car il y a des stories sur le sujet à affiner. Mais Laurent a dû oublier, il n'est pas là ! Nicolas l'appelle et apprend qu'il a une urgence. Il négocie sa venue pour un quart d'heure. On change un peu l'ordre des activités de la réunion pour saisir le créneau, c'est important qu'il soit là. Finalement, l'affinage se passe bien, il y a suffisamment de stories prêtes, Nicolas en compte 10.

Après la réunion, il reste avec Céline le PO pour mettre à jour le plan de release, qui a été pas mal touché par le travail d'affinage. Mais il est appelé par Sébastien qui lui annonce que le serveur de développement est en rade. Il laisse Céline finir et file voir Sébastien. Bon, pas trop grave, il suffisait de relancer le serveur.

Il a un peu de temps avant sa réunion pour analyser les raisons profondes du gros bug de la semaine dernière, alors il passe voir l'essaim qui s'occupe de la story « Modérer les photos de chien ». Il aide en passant deux vérifications de sa définition de fini. La story va être finie ce soir !

Il anime la discussion sur le gros bug, en proposant les 5 pourquoi pour remonter à l'origine du problème. Mmm, il semble qu'il faudrait ajouter une règle de codage.

Lors de la mêlée du matin, il a deviné qu'Émilie avait des soucis. Il va la voir, avant qu'elle parte. OK, il arrive à comprendre qu'elle est en conflit avec David, il ira lui parler demain. Faudra qu'il pense à proposer un niko-niko à la prochaine rétrospective pour, peut-être, anticiper ce genre de situation.

Avant de partir, il consulte ses messages et voit une demande de Kevin qui voudrait emmener Julien, dès demain et pendant 2 jours faire des démos chez des clients. Après une discussion franche, il dit non, cela remettrait en cause l'objectif du sprint.

5.5. Le SM sur le terrain

Des difficultés peuvent apparaître quand le ScrumMaster remplit mal son rôle, par exemple s'il ne fait pas confiance aux membres de l'équipe et décide à leur place.

5.5.1. Tourner dans le rôle

Dans une équipe aguerrie, la personne qui joue le rôle de ScrumMaster peut tourner : à chaque sprint, ou au bout de quelques sprints, on change.

ScrumMaster devient alors un rôle dynamique, cela évite à une personne qui n'est pas faite pour cela de s'installer dans la routine ou de retomber dans des travers de chef. Cela permet aussi d'apprendre, en voyant les attitudes des autres.

On fait tourner le scrummaster dans l'équipe !

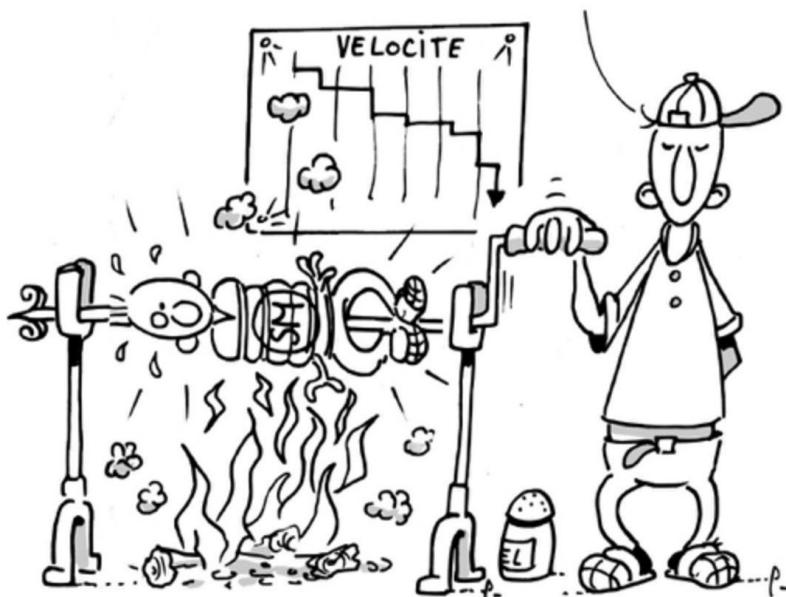


Fig. 5.3 Faire tourner un SM trop autoritaire

Cela s'est produit dans mes projets avec des étudiants. Tous les membres d'une équipe d'étudiants sont dans la même classe et ont, *a priori*, la même expérience. Aucun d'entre eux n'a jamais été ScrumMaster auparavant, ni chef de projet d'ailleurs. Le choix du ScrumMaster est fait par l'équipe, les enseignants n'interviennent pas. Lorsque le projet avance, il est proposé, si l'équipe ne le demande pas elle-même, que ce rôle soit tournant. Le choix est laissé à l'appréciation de l'équipe.

5.5.2. Parfaire sa connaissance de Scrum

Être un bon ScrumMaster nécessite une culture agile et une maîtrise de Scrum. Cela s'apprend d'abord en appliquant, bien sûr, mais aussi en lisant des livres ou des articles. La participation à des conférences où sont présentés des retours d'expérience est

particulièrement enrichissante. Il existe des groupes d'utilisateurs, comme le *Scrum User Group* français^[1] et de nombreux autres au niveau local ou régional.

Dans certaines sociétés, généralement des petites, la personne qui devient ScrumMaster était située, dans la hiérarchie, sous l'autorité de celle qui prend le rôle de Product Owner. Une bonne connaissance de Scrum lui permettra de s'affirmer, ce qui aura pour effet de limiter un pouvoir excessif du Product Owner.

Au-delà de la maîtrise de Scrum et de l'état d'esprit, devenir un bon ScrumMaster nécessite des compétences particulières qu'une formation aide à acquérir.

On ne conseillera pas ici les formations dédiées au ScrumMaster mettant en avant les certifications contestées des organismes américains, mais plutôt une formation de toute l'équipe. Le SM pourra acquérir des compléments de formation plus tard, après une première expérience.

Dans certaines situations, la meilleure solution est d'être accompagné par un expert Scrum dans sa mise en œuvre sur le projet. C'est particulièrement important pour de grandes organisations dans lesquelles la culture traditionnelle des projets est fortement marquée. Elles semblent résister de façon coriace au changement et le *coaching* des ScrumMasters y est indispensable dans les premières expériences de Scrum.

5.5.3. Savoir changer de posture

Lorsqu'un ScrumMaster s'aperçoit qu'il est moins indispensable à l'équipe, c'est probablement qu'il a réussi, il doit changer de posture.

Comme le dit Charles Piaget dans le film *Les Lip*^[2] :

« *Un leader sait qu'il a réussi quand on n'a plus besoin de lui ou en tout cas quand sa voix ne compte que pour un, comme celle de tout le monde dans le groupe.* »

C'est sûrement plus facile à mettre en place dans le développement de logiciel que dans la production de montres. Le paradoxe est que le ScrumMaster qui a réussi devient inutile dans son équipe...

Le rôle de ScrumMaster évolue avec la maturité de l'équipe : au début, il passe beaucoup de temps à apprendre Scrum à l'équipe, ensuite il a plus un rôle de conseiller (expert, mentor, coach).

Certains auront envie d'aller plus loin, en apprenant des techniques de coaching d'équipe [Messenger, Coacher].

Bien commencer

La valeur à partager avec l'équipe	Le respect, qui consiste à penser que chacun fait de son mieux dans l'équipe.
De mauvais signes	Le ScrumMaster fait le ménage et les courses. C'est lui seul qui écrit tous les Post-it®.
Par quoi démarrer	S'assurer qu'il y a des Post-it® et des bières.
Une lecture pour tous	<i>Coaching Agile</i> [Davies, <i>Coaching Agile</i>], qui donne de très bons conseils à un ScrumMaster, même si le livre est plutôt orienté XP que Scrum.

À retenir

Le ScrumMaster ne gère pas des ressources interchangeables, il guide les femmes et les hommes de l'équipe. Son rôle essentiel est de les faire progresser collectivement pour la réussite des sprints et des releases.

Les méthodes agiles reprennent l'idée d'organisation sans hiérarchie autoritaire : on y parle d'équipe investie avec le pouvoir et l'autorité pour faire ce qu'elle a à faire ou qui s'organise par elle-même. C'est une des différences majeures avec les méthodes traditionnelles. Elle est mise en pratique avec le ScrumMaster, qui n'est pas un chef mais un facilitateur.

Il agit en veillant à la mise en application de Scrum, en faisant en sorte que les événements aient lieu en étant alignés avec ses valeurs et ses principes, en encourageant l'équipe à apprendre et à progresser, en faisant en sorte d'éliminer les obstacles qui freinent l'équipe, et d'éviter les perturbations venant de l'extérieur.

Références :

- ☞ Rachel Davies & Liz Sedley, *Coaching Agile*, 2009, traduit en français par Fabrice Aimetti.
<http://ayeba.fr/coaching-agile/>
- ☞ Dominique Dupagne, *La revanche du rameur*, 2012.
<http://www.larevanchedurameur.com/>
- ☞ Véronique Messager, *Coacher une équipe agile*, Eyrolles, 2012.

Notes

- [1] Pour en savoir plus : www.frenchsug.org.
- [2] Voir http://fr.wikipedia.org/wiki/Les_Lip,_l%27imagination_au_pouvoir

Structurer le backlog

Après avoir décidé de lancer le développement d'un produit, la difficulté fondamentale est de transformer la vision de départ en quelque chose d'exploitable par l'équipe de développement.

Dans les projets traditionnels, la « spécification » se fait entièrement au début du projet et se concrétise dans un document qui décrit ce que va faire le produit, quelles sont les fonctions attendues et quel est le comportement souhaité. Ce document devient volumineux car on se dit qu'il faut tout écrire et tout détailler pour ne rien oublier afin que les développeurs sachent bien ce qu'il y a à faire.

Dans ma carrière, j'ai passé du temps à rédiger de gros documents de spécification. Je me souviens qu'à une époque où je travaillais dans une entreprise des télécoms, j'ai écrit le document de spécification^[1] externe du système. Je faisais de mon mieux pour imaginer le comportement du produit. Comme il s'agissait de téléphonie, j'étais un utilisateur potentiel du système et cela facilitait mes réflexions. Je publiais des versions fréquentes de ce document en essayant d'avoir du feedback du marketing. Même si les retours étaient plutôt rares, j'avançais bien dans la rédaction du document.

Cela m'a apporté beaucoup sur la connaissance du produit. Quand il a fallu transmettre cette connaissance aux développeurs de l'équipe, cela a mieux fonctionné avec des discussions régulières que par leur lecture, plutôt modérée, du document. Ce sont ces échanges qui ont permis à l'équipe d'avancer : le produit a été développé et, après quelques mois, commençait à fonctionner.

Le gros document, dont j'étais plutôt fier, ne leur a pas vraiment servi. Il n'a pas convenu non plus, un peu plus tard, à l'équipe chargée des tests fonctionnels. C'est vrai qu'il n'était pas facile de rentrer dans les deux cents pages pour quelqu'un qui ne connaissait pas bien le domaine. Et bien sûr, il n'était plus à jour.

Dans le domaine du logiciel, le constat est qu'un gros document de spécification élaboré au début par une seule personne est difficile à maintenir, qu'il n'est pas très utile pour le développement et encore moins pour les tests.

Avec Scrum, la démarche est fondamentalement différente : les spécifications émergent progressivement par une collaboration continue entre les membres de l'équipe et le Product Owner, et grâce aux feedbacks réguliers donnés par les parties prenantes.

L'outil de collecte et de partage s'appelle le **backlog**.

6.1. Un outil essentiel pour l'équipe

Mais n'y aurait-il pas un mot pour traduire *backlog* en français ?

6.1.1. Vocabulaire

Sur le premier projet que j'ai accompagné dans une transition à Scrum, c'est le terme « référentiel des exigences » qui a été utilisé. Mais, si le backlog peut être considéré comme une référence, ce ne sont pas des exigences qu'on y met. Dans une approche agile, quelqu'un n'exige pas quelque chose de quelqu'un d'autre, on converse autour d'« histoires », pour mieux se comprendre.

Nos cousins québécois, qui sont de fervents adeptes de la francisation des mots issus de l'anglais, ont traduit par « carnet », mais cela n'a pas percé chez nous. L'usage courant est de ne pas traduire backlog.

Cela n'est pas sans créer quelques confusions : certains ont du mal avec le backlog, le prononçant « blaquélogue ». J'ai même entendu un « blackdog ». Quelques-uns persistent à dire « la backlog ».

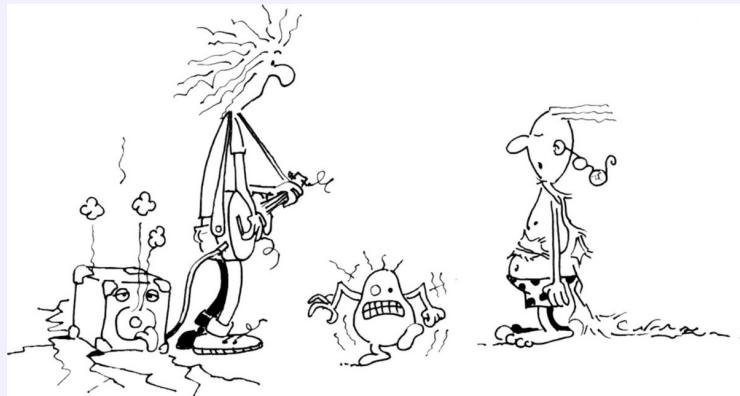


Fig. 6.1 « *J'ai dit faire péter le backlog, pas répéter black dog !* » remarque d'un ScrumMaster à un développeur fan de Led Zeppelin.

6.1.2. Une liste unique

En première approche, un backlog est simplement une liste ordonnée des choses à faire par l'équipe.

L'équipe trouve dans le backlog ce qui est d'ordinaire dispersé dans plusieurs documents ou outils. Le fait qu'il soit unique va faciliter son partage et son maintien en bon état.

J'ai commencé le chapitre en parlant de spécification. Le backlog n'est pas seulement un outil de collecte de ce que doit faire le produit, comme une spécification. C'est aussi un outil de pilotage de projet : ce qui est fini y est enregistré et permet de suivre l'avancement.

6.1.3. Une liste publique

Certes, le Product Owner est le plus actif dans l'élaboration et le maintien du backlog ; cependant, c'est l'outil de toute l'équipe.

Le backlog est également visible des parties prenantes qui sont intéressées par le développement du produit : clients, utilisateurs, managers, marketing, opérations, etc.

Tout ce monde y a accès, pour favoriser la transparence et faciliter le feedback, qui se concrétise par l'ajout de nouvelles idées.

6.1.4. Une liste ordonnée

Le backlog indique l'ordre dans lequel seront réalisés les éléments qu'il contient.

Priorité ordinaire

Dans un backlog, les éléments sont rangés selon l'ordre envisagé pour leur réalisation. Cette notion de priorité ordinaire, qui n'est pas usuelle dans les documents de spécification, est adaptée au développement itératif : l'ordre est celui de réalisation.

Dire que A est plus prioritaire que B signifie que A sera réalisé avant B.

Critères pour définir l'ordre

Les méthodes agiles ont pour but de maximiser la valeur. La notion de valeur va donc être un critère majeur pour définir l'ordre.

La valeur donne une idée de combien ça va « rapporter » aux parties prenantes. Combien ça va « coûter » en développement est aussi un critère important.

Pour définir l'ordre d'un élément par rapport à un autre, l'idée est de maximiser la valeur rapportée au coût, en tenant compte des dépendances.

6.1.5. Une liste vivante

Le backlog suit la vie du produit qu'il décrit, il évolue avec lui ; il peut donc durer très longtemps, utilisé sur plusieurs releases successives.

Émergence progressive

J'ai consulté de nombreuses spécifications dans différents domaines du développement de logiciel. Le constat a toujours été le même : il est impossible de tout connaître dès le début d'un projet. En réfléchissant longtemps et en essayant d'imaginer les situations dans lesquelles se trouveront les utilisateurs, on peut bien sûr découvrir un bon nombre de fonctions significatives, mais il en existera toujours qui émergeront plus tard.

Seul le feedback des utilisateurs sur une version opérationnelle permet de savoir ce qu'ils attendent réellement. Le contenu du backlog émerge continuellement.

L'émergence est un concept fondamental de l'agilité. Plutôt que d'essayer de figer les besoins, on favorise les nouvelles idées.

Changements permanents

Dans un développement agile, le changement est possible et même encouragé. Il ne coûte presque rien, tant qu'il porte sur un élément pour lequel on a fait peu d'effort, en regard de la valeur qu'il peut apporter.

Le backlog n'est pas figé, il n'est jamais complet ou fini tant que vit le produit. Il est élaboré dans une forme initiale pendant le sprint zéro, puis il évolue constamment : des éléments sont ajoutés, d'autres sont supprimés, certains sont décomposés et l'ordre ajusté.

Ce changement continual n'a pas d'impact immédiat sur les réalisations en cours ; pendant un sprint, la partie du backlog sur laquelle l'équipe travaille n'est, en principe, pas modifiable

6.2. Hiérarchie des éléments de backlog

Scrum n'impose pas de règles pour identifier les éléments du backlog. Certains les appellent toujours « PBI », pour *Product Backlog Item*.

Après plus de dix années de pratique du backlog, j'en suis arrivé à conserver trois types d'éléments : la story, l'épic et la feature.

6.2.1. La story

La pratique des user stories, venant de XP, est désormais fortement associée à Scrum.

Une *user story* est un petit morceau de fonctionnalité visible d'un utilisateur et qui peut être développé en un sprint.

Une user story de Peetic : pour un propriétaire, avoir la possibilité d'inscrire son basset hound à une exposition canine.

Cependant, dans un backlog, toutes les histoires (*stories*) ne sont pas relatives à des utilisateurs (*user*). C'est pourquoi j'utilise le terme général de **story** pour parler d'un élément du backlog.

Quand elle est créée, une story est simplement nommée, éventuellement avec un verbe et un complément. Ensuite, au cours de sa vie, des attributs vont être ajoutés, par exemple : son rang dans le backlog, son état, son type, et d'autres informations que l'équipe jugera intéressantes.

Dans la vie de la story, le moment structurant est le sprint, c'est pourquoi son contenu est vérifié au départ et à l'arrivée de ce sprint.

- Pour que la story prenne le départ avec de bonnes chances d'arriver au bout du sprint, l'équipe doit s'assurer qu'elle est prête. Les vérifications à faire sont listées dans la *définition de prêt*.
- Une story commencée est normalement finie avant la fin du sprint. Sa *définition de fini* liste les points à vérifier pour s'en assurer.

Nous approfondirons les notions de *prêt* et de *fini* dans les chapitres suivants.

6.2.2. L'epic

L'epic (ou story épique) est une story dont l'équipe considère qu'elle ne peut pas entrer dans un sprint en l'état. Elle ne peut pas passer au statut « prête ».

La raison pour laquelle une story est épique, c'est qu'elle est trop grosse. C'est aussi le cas quand elle est mal connue, car on ne sait pas encore ce qu'elle cache. Une story épique doit donc être décomposée ou étudiée : elle a besoin d'affinage.

Une épique de Peetic : gérer les séances de coaching du chien, story épique, pourra se décomposer en plusieurs stories « s'inscrire », « accepter l'inscription », « de désinscrire », etc.

La story épique cohabite avec la story élémentaire dans le backlog. Une fois décomposée, elle disparaît. Son usage est optionnel, il est utile pour identifier clairement le travail de décomposition à faire et pour limiter le nombre d'éléments.

6.2.3. La feature

Une *feature* est un service ou une fonction d'un produit dont l'énoncé est clair pour les parties prenantes ; une feature contribue à un impact et se décompose en stories.

Une feature de Peetic : coaching en ligne pour animaux.

Une feature sera réalisée par plusieurs stories. On pourrait la mettre dans le même backlog que les stories, mais, en général, il est préférable de les mettre ailleurs, pour y voir plus clair.

Pour éviter la confusion possible avec un deuxième backlog, je vais parler dans la suite du livre du « tableau de features ».

Une feature n'est pas rythmée par le sprint comme la story. Elle peut être non finie à la fin d'un sprint. En revanche, elle doit être finie à la fin de la release, au plus tard. C'est une notion essentielle qui nous accompagnera.

6.3. Types de stories

L'idée de ne mettre dans le backlog que des user stories ne résiste pas à la réalité du terrain : la vocation du backlog étant de collecter tous les travaux significatifs de l'équipe, il serait dangereux de se limiter uniquement à ce qui est destiné aux utilisateurs finaux et de ne pas rendre visibles des travaux destinés à d'autres parties prenantes ou à l'équipe.

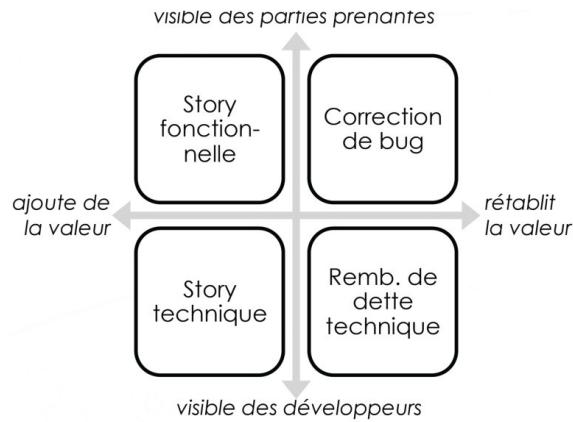


Fig. 6.2 *Les grands types de story*

Ma présentation des types de story reprend une proposition de Philippe Kruchten[Kruchten, Color]. Elle présente deux axes :

- la valeur, selon que la story en ajoute ou rétablit celle perdue ;
- la visibilité, selon à qui elle est destinée.

Cette classification sert d'abord à ne pas oublier de travail. Elle est utile, avec une classification plus fine, dans la *définition de prêt et de fini*.

6.3.1. Story fonctionnelle

La plupart des stories fonctionnelles sont des user stories : elles s'adressent au sous-ensemble des parties prenantes que sont les utilisateurs, en leur fournissant de la valeur.

Exemple Peetic : pour notre site de rencontres d'animaux, « Ajouter une photo de son animal » est une user story.

Cette story est réalisée avec du code. Il peut y avoir d'autres stories, ne contenant pas de code, qui sont visibles des parties prenantes.

Exemple : la réalisation par l'équipe d'une capture vidéo montrant aux utilisateurs comment ajouter une photo de leur animal.

Ces stories permettent d'apprendre sur le produit grâce au feedback.

6.3.2. Correction de bug

Un bug enlève de la valeur au produit, et la story sur laquelle il porte n'a plus toute celle promise. C'est sa correction qui va rétablir la valeur.

La notion de bug est assez subtile avec Scrum. Il ne faudrait pas penser qu'on stocke tous les bugs dans le backlog. D'abord, on s'efforce de détecter et corriger les erreurs introduites le plus vite possible, pendant le sprint. Et donc cela ne passe pas par le backlog. N'est pas considéré non plus comme un bug l'ajout d'une nouvelle condition d'acceptation : c'est une nouvelle story.

Les bugs dont il est question ici, ceux qui entrent dans le backlog, correspondent à des situations où on se rend compte qu'une story, déclarée finie dans un sprint précédent, présente une imperfection qui nuit à son usage. En principe, il ne s'agit que de bugs mineurs.

Exemple de correction de bug : ajouter un retour à la ligne automatique sur le texte de la légende de la photo de l'animal, qui déborde sur la colonne d'à côté.

6.3.3. Story technique

Il s'agit des travaux qui apportent de la valeur à l'équipe mais qui ne sont pas visibles par les parties prenantes.

Attention, dans les *user stories*, il y a toujours des travaux techniques (conception, code) et c'est très bien. Il ne s'agit pas de faire une story technique pour y mettre les travaux techniques de chaque story.

La story technique se justifie dans plusieurs situations :

- il y a une étude technique à faire pour prendre une décision sur la façon de réaliser une user story ;
- il existe un risque qu'on veut lever avant la réalisation d'une user story ;
- pour des travaux visant à améliorer la qualité ou la façon dont l'équipe développe : infrastructure, logistique, nouvel outil, formation, etc.

Exemple de story technique : étudier une solution de stockage pour les photos et vidéos d'animaux (dans le cas où ce travail ne peut pas être inclus dans la story Ajouter une photo ou une vidéo).

Dans les premiers sprints d'un nouveau développement avec une nouvelle technologie, la réalisation de stories, dont des « techniques », a pour objectif de diminuer le risque, pour ensuite, dans les sprints suivants, ajouter régulièrement de la « valeur métier » avec les user stories.

6.3.4. Remboursement de dette technique

La dette technique est le cauchemar du développement de logiciel. Tout le monde en a, mais peu en sont vraiment conscients. Elle ne se voit pas de l'extérieur de l'équipe ; les développeurs, s'ils s'en aperçoivent, ont du mal à la faire comprendre aux autres.



Fig. 6.3 Pas facile de convaincre un PO qu'il faut réduire la dette

La dette technique peut être volontaire, dans le cas où est privilégié le feedback rapide. On ajoute alors une entrée dans le backlog pour ne pas oublier de reprendre la partie volontairement moins finie.

Avec Scrum, on cherche d'abord à l'éviter. Mais, s'il y en a, il est bien de la rembourser le plus tôt possible, car les intérêts s'accumulent, comme pour une dette financière. Pour cela, la solution est de faire apparaître les travaux de remboursement de façon explicite, c'est-à-dire dans le backlog.

Une fois le plus gros de la dette remboursée, il faut prendre les mesures pour ne plus en réintroduire, c'est l'objectif de la définition de fini.

Exemple de remboursement de dette technique : remanier le code du composant de téléchargement.

6.3.5. Features et types de story

Lors de la décomposition d'une feature, il est usuel de trouver des stories fonctionnelles, user stories et autres (us), mais aussi une ou plusieurs stories techniques (tec), et une story spéciale de finition de feature.

Une feature est décomposée en différents types de story.

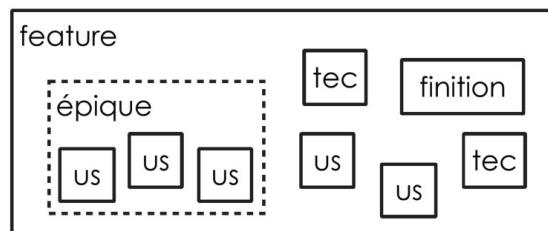


Fig. 6.4 Une feature décomposée en stories

Je conseille de conserver l'association entre la feature et les stories qu'elle contient en trouvant un moyen de la visualiser facilement, par exemple un code couleur.

6.4. Parties du backlog

Quand on débute avec Scrum, le backlog est bien souvent monolithique. On voit, en particulier chez ceux qui utilisent des tableurs, des backlogs avec des dizaines et des dizaines d'éléments mélangés.

C'est vrai qu'en prenant au pied de la lettre le backlog comme une liste d'éléments, on peut se dire que les lignes d'un tableau feront l'affaire. Mais, et même si on est expert dans l'utilisation des filtres du tableur, cela risque de ne pas faire l'affaire bien longtemps. C'est peu praticable. C'est souvent trop gros. Et cela se partage mal.

Le backlog est d'abord un outil de communication.

6.4.1. Workflow de la story

En 2001, Ron Jeffries définissait la vie de la story avec trois phases : la carte comme moyen de l'identifier, puis une conversation et enfin une confirmation. Cela est connu comme les « 3C »^[2].

En fait, l'équipe déroule deux cycles de conversation et confirmation : un premier pour obtenir une story prête, un second pour qu'elle soit finie :

1. Un jour, quelqu'un a une idée de story et la note sur une **Carte** (maintenant on utilise plutôt un Post-it®).
2. Le Product Owner et l'équipe affinent cette story, afin qu'elle puisse être réalisée en un sprint, au cours d'une **Conversation**.
3. L'équipe apporte sa **Confirmation** qu'elle est prête.
4. L'équipe réalise la story pendant un sprint, en tenant une nouvelle **Conversation** avec le PO, sur son acceptation.
5. Le Product Owner apporte sa **Confirmation** qu'elle est finie.

Finalement, nous avons donc « 5C » dans la vie de la story, avec ces deux grandes phases de travail collectif basées sur des conversations : la **réalisation** pendant un sprint, bien connue, et l'**affinage** dans des sprints antérieurs, moins connu.

Entre ces deux travaux non consécutifs, la story est en attente. Elle est prête, au sens où elle peut être réalisée dans un prochain sprint.

En se basant sur ces « 5C », on peut représenter le workflow de la story (figure 6.5).



Fig. 6.5 Cycle de vie d'une story

On ne traite pas de la même façon une story au stade de l'idée qu'une story prête. L'affinage nécessite une identification claire des stories.

Cela pousse à ranger les stories dans des dépôts correspondant à leur état. Nous les appelons des **bacs**, en référence au backlog.

6.4.2. Bac à sable pour les idées

Le bac à sable est en quelque sorte l'antichambre du backlog. C'est la boîte d'entrée dans laquelle tout le monde, y compris les développeurs et les parties prenantes, propose des stories.

Plus exactement, on met dans le bac à sable une idée d'un « truc » à ajouter au produit, qui pourra éventuellement devenir une story.

La durée de séjour est généralement courte, le temps pour le PO de comprendre la proposition et de la traiter. Le bac à sable n'est ni ordonné ni estimé. Une story dans le bac à sable possède peu d'attributs. Des conversations peuvent être nécessaires entre le PO et le demandeur s'il est nécessaire de clarifier sa demande.

Séparer le bac à sable du reste du backlog permet à tous de faire des propositions, et au PO de les traiter. Les méthodes agiles favorisent le feedback, le bac à sable en est le réceptacle initial.

6.4.3. Bac d'affinage

C'est l'endroit où l'équipe affine les stories de la release courante, avant de les réaliser dans un sprint. Comme dans une cave pour les fromages, l'affinage consiste à entourer de soins délicats et attentifs les stories, à les entretenir régulièrement, à les affiner. Le bac a une allure typique (figure 6.6) :

- Une story prioritaire va être bientôt prête. Elle est de petite taille.
- Ce qui est au milieu du bac d'affinage ne sera développé que dans quelques sprints. On y trouve des stories épiques de taille moyenne.
- Ce qui est moins prioritaire sera développé encore plus tard. On y trouve des *épiques* de plus grande taille qui seront décomposées ultérieurement.

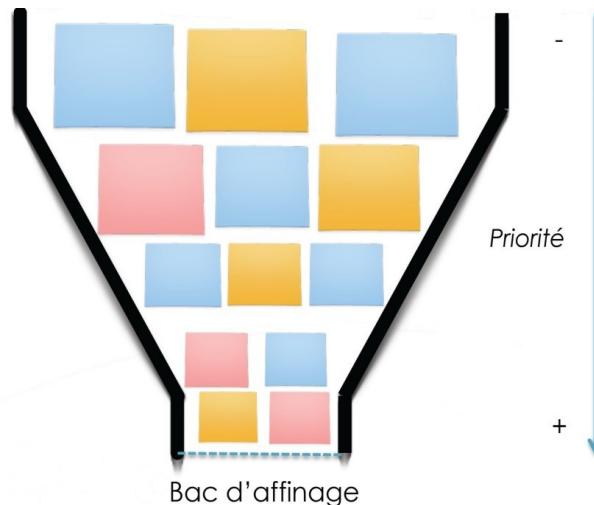


Fig. 6.6 Allure du bac d'affinage

Le bac d'affinage est en forme d'entonnoir : les plus grosses stories ne peuvent pas s'écouler, seules les petites peuvent passer dans le bac suivant.

Le bac d'affinage définit le périmètre de la release : il est vide à la fin d'une release et alimenté au début de la suivante.

Le bac d'affinage permet de se concentrer sur la release courante.

6.4.4. Bac de départ pour les stories prêtes

Puisque Scrum a choisi la métaphore de la course (le sprint), poursuivons avec l'idée qu'avant d'être en course dans le sprint, les stories se trouvent dans les starting-blocks, que nous allons appeler le bac de départ.

Comme dans les vrais sprints avec le coup de feu du starter, le démarrage se fera sur le top de lancement : lors de la planification du sprint, des stories sélectionnées prennent le départ du sprint.

Ce bac est en quelque sorte une file d'attente dans laquelle la story n'évolue pas, ou peu, depuis sa sortie du bac d'affinage. Le nombre d'éléments ne devrait pas dépasser l'équivalent d'un peu plus d'un sprint.

Le bac de départ permet de limiter les risques pour le prochain sprint.

Les stories en développement effectuent leur sprint. Elles sont « en course ». On suit leur parcours, nous y reviendrons dans les événements du sprint.

6.4.5. Bac d'arrivée pour les stories finies

C'est l'endroit où l'on dépose les stories finies. On n'attend pas la fin du sprint pour déclarer une story finie, c'est fait au fil de l'eau.

On peut considérer que le bac d'arrivée correspond à une file d'attente des stories pour la démonstration de fin de sprint, et qu'ensuite il est vidé.

Mais le mieux est d'y conserver les stories finies jusqu'au moment où elles seront déployées, en général à la fin de la release. Dans ce cas, on y range les stories par sprint.

Ce bac permet aux parties prenantes de visualiser les résultats des travaux de l'équipe. C'est aussi le moyen de conserver l'historique de la release.

Avec ces quatre bacs plus le sprint, l'équipe (en particulier le Product Owner) dispose de dépôts facilitant la gestion d'un backlog, lui permettant :

- d'y voir clair, chaque bac ayant un objectif différent ;
- d'avoir un usage adapté à chaque objectif ;
- de ne pas être perturbée par un stock de stories.

Ce chapitre présente une vue statique des bacs, nous verrons dans le prochain la vue dynamique, quand les stories passent de bac en bac.

6.5. Tableau de features

6.5.1. Workflow de la feature

Le workflow d'une story qui nous a amenés aux bacs est imposé par la nature des sprints dans Scrum.

Les notions de prêt et de fini découpent la vie de la story en deux grandes parties : l'affinage et la réalisation. C'est, me semble-t-il, adapté à toutes les situations ; tout au plus il serait possible de décomposer l'affinage en plusieurs activités.

En revanche, le workflow d'une feature dépend largement du contexte. Il est influencé par la place donnée à Scrum dans la chaîne de valeur de la feature : en amont, par le processus de décision de lancer la feature, en aval par le processus de déploiement.

La vie d'une feature, entre le moment où c'est encore une opportunité et celui où elle est utilisée, dépend donc du contexte. Les états significatifs peuvent varier selon le type de déploiement, la taille du produit, le rythme des sprints et des releases, les indicateurs à produire. Cependant, j'ai trouvé un schéma qui convenait à de nombreuses situations, en tout cas qui pouvait servir de point de départ :

- **Opportunité** : c'est un peu l'équivalent du bac à sable. Chaque idée de feature représente une opportunité pour le produit.
- **Prévu pour la release** : l'étude d'opportunité a abouti, la feature est prévue pour la release en cours.
- **En affinage** : on commence à travailler sur cette feature. Elle est « copiée » dans le bac d'affinage, sous forme d'épique, pour être décomposée en stories.
- **Minimisé** : l'affinage des stories a permis d'identifier le minimum pour que la feature ait du sens et apporte de la valeur. Elle est devenue une MMF (*Minimal*

Marketable Feature).

- **Testé** : la feature est testée (avec ses stories, finies elles aussi) et tourne, généralement dans un environnement de pré-production. C'est ce qu'on appelle une RTF (*Running Tested Feature*).
- **Déployé** : elle est à disposition des utilisateurs, qui sont sollicités pour fournir du feedback.

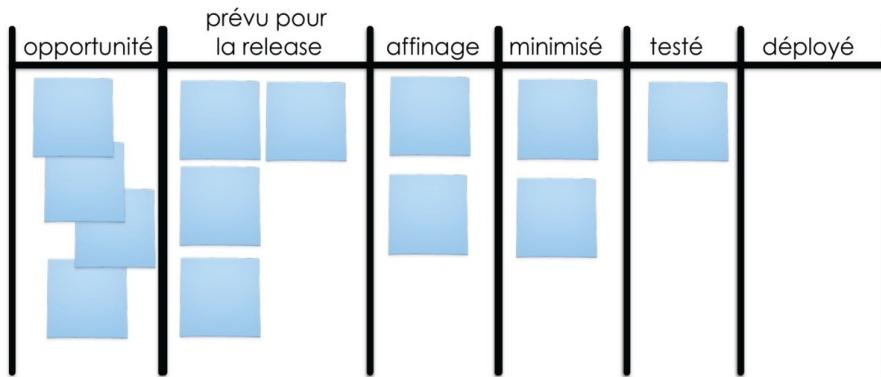


Fig. 6.7 Tableau de features

6.5.2. Priorité des features

Les expérimentations montrent que la notion de valeur s'applique bien mieux à la feature qu'à la story. La valeur est le critère essentiel pour définir la priorité des features, permettant de les ordonner dans chaque colonne.

6.6. Le backlog sur le terrain

6.6.1. Migrer trop d'éléments

Quand une équipe démarre avec Scrum, il peut exister des documents de spécification traditionnelle déjà rédigés, avec peut-être des centaines d'exigences identifiées et numérotées. Il est tentant d'en faire des entrées du backlog.

Les mettre dans le backlog sans discernement le rendrait ingérable. D'une part, parce qu'il y aurait trop d'éléments, d'autre part, parce qu'il faut structurer les éléments introduits dans le backlog.

La solution est alors d'analyser le document pour vérifier sa compatibilité avec une approche agile. Si ce n'est pas le cas, il vaut mieux s'en servir comme point de départ pour partir sur une identification des stories puis s'en débarrasser.

D'un autre côté, on trouve parfois de grands stocks de bugs résiduels que l'équipe a du mal à éliminer. C'est d'ailleurs souvent pour essayer de régler le problème que l'équipe passe à Scrum.

Seulement, transférer des centaines de bugs d'un « bugtracker^[3] » dans le backlog n'est pas une bonne idée, pour les mêmes raisons de taille : un tri préalable s'impose avant de les inclure.

6.6.2. Éviter d'avoir plusieurs backlogs pour une équipe

L'idée historique est d'avoir un backlog par produit, une autre pratique est d'avoir une équipe pour développer un produit, avec la recommandation que l'équipe est à temps plein sur un seul projet. Mais de nombreuses organisations passant à Scrum sont dans un schéma où une seule équipe travaille sur plusieurs projets en même temps.

Bien souvent elles abordent la transition à Scrum en faisant un backlog par projet. Ce n'est pas la bonne approche : les problèmes de priorité ne seront pas résolus, sans compter les autres inconvénients du multiprojet. La solution est de rester dans le cadre : une équipe – un backlog.

6.6.3. Communiquer avec des conversations

Tout n'est pas écrit dans le backlog : une story est l'objet de discussions avant et pendant le sprint où elle est développée. La communication directe permet de la faire passer de l'esprit du Product Owner à celui du développeur, tout en laissant à ce dernier une marge de manœuvre pour sa réalisation.

La communication en face à face est recommandée.

6.6.4. Tenir compte des dépendances

On dit qu'une story B dépend d'une story A si B ne peut être développée que lorsque A est finie.

L'ordre en tient compte : la story A sera plus prioritaire que B. Si on développe B avant A, on a dérogé à la dépendance, ce qui impliquerait des travaux supplémentaires pour l'acceptation de B.

Exemple Peetic : la story « Ajouter une photo sur la fiche animal » dépend de la story « Créer la fiche animal ».

Il n'est en général pas nécessaire de mentionner explicitement les relations de dépendances dans un backlog : le PO les prend en compte au jour le jour, chaque fois que l'ordre est revu.

De leur côté, les développeurs peuvent faire remonter des dépendances techniques subtiles entre des stories.

Une autre dépendance qui peut jouer sur l'ordre des stories, c'est celle qui porte sur l'équipe. Quand elle n'est pas totalement pluridisciplinaire, il arrive que des stories soient décalées pour tenir compte de la disponibilité d'un expert nécessaire à son accomplissement. Attention dans ce cas à ne pas perdre de vue le produit et plus précisément la finition d'une feature.

Bien commencer

La question à se poser	Est-ce que le backlog est utilisé par toute l'équipe ? Le backlog comporte plus de 100 éléments. Le backlog n'est pas ordonné.
De mauvais signes	Le backlog contient des tâches qui n'apportent pas de valeur. Le backlog est enfoui dans un tableau auquel personne n'accède.
Par quoi démarrer	Représenter le backlog en bacs sur le mur, avec des Post-it®. Faire un tableau de features
Une chanson pour tous	<i>Black Dog</i> de Led Zeppelin, sur l'album IV.

À retenir

Le backlog contient la liste des réalisations de l'équipe, les stories. C'est l'élément pivot d'un développement avec Scrum, qui permet de définir le produit et de faire la planification.

Cette liste unique des choses à faire, ordonnée, est au cœur de la mécanique de mise en œuvre de Scrum lors des sprints. Le backlog de produit contribue beaucoup à l'élégance et à la simplicité du cadre de développement que constitue Scrum.

La structure du backlog se base sur trois notions : la hiérarchie des éléments (feature, story), la vie de ces éléments représentée avec les bacs et leur type, selon les parties prenantes auxquelles ils apportent de la valeur.

Références :

- ☞ Philippe Kruchten, *What Color is your Backlog ?* InfoQ.
<http://www.infoq.com/news/2010/05/what-color-backlog>
- ☞ Jeff Patton, *Story Mapping*, Dunod, 2015.
- ☞ Dan Rawsthorne, Doug Shimp, *Exploring Scrum, The Fundamentals*, CreateSpace Independent Publishing Platform, 2011.

Notes

[1] Il existait différentes appellations de « spécification » : fonctionnelle, externe, interne, etc. En anglais on utilise le terme *requirement*, comme dans *Software Requirement Specification*.

[2] Article traduit par Fabrice Aimetti : <http://wiki.ayeba.fr/XP%2C+l%27essentiel+-+Carte%2C+Conversation%2C+Confirmation>.

[3] Type d'outil couramment utilisé pour le suivi des bugs.

Affiner le backlog

On constate que de nombreuses équipes ne tiennent pas leur engagement ou plus largement n'arrivent pas à finir autant de stories qu'elles voudraient à la fin du sprint. En remontant à l'origine du problème, on s'aperçoit que ces équipes embarquent dans le sprint des stories qu'elles connaissent mal ou qui sont trop grosses. Elles se rendent compte brutalement, pendant le sprint, de l'impossibilité de les finir. La raison est que ces stories n'avaient pas été affinées.

L'affinage consiste à maintenir le backlog prêt pour augmenter les chances de succès des futurs sprints.

En anglais, le terme employé a longtemps été *grooming*. Je l'ai d'abord traduit par bichonner. Dans la première édition ce livre, je disais « *le PO bichonne son backlog* ». Dans l'édition trois, je suis passé au *PO qui cultive son backlog*, j'ai parlé de *bac de culture*. Entre-temps, dans le petit *Guide Scrum* [Sutherland, *Scrum Guide*], version 2013, *grooming* est devenu *refinement*. Puisque c'est le terme officiel en anglais, je vais utiliser sa traduction.

Attention, pas raffinement ! Pour un backlog raffiné, il suffirait de le décorer avec de jolis Post-it®. Non, la bonne traduction est affinage.

L'affinage est une activité continue, dont le résultat est le reflet de l'émergence des stories, possible et même encouragée tout au long du développement du produit.

7.1. Définition de prêt

7.1.1. Prêt pour le backlog

L'objectif principal de l'affinage est d'avoir suffisamment de stories prêtes pour le prochain sprint.

« Suffisamment » varie selon le contexte et dans le temps : au départ d'un sprint, ce nombre correspond au moins au total des stories habituellement finies pendant un sprint. Pour le prévoir, on se basera sur les mesures des sprints précédents.

Pour savoir si le backlog est « prêt », on regardera donc si la liste des stories prêtes est assez longue, et bien ordonnée.

Exemple : pour Peetic, la moyenne de stories finies dans un sprint est de 10. En fin de sprint, on devrait trouver 12 stories prêtes et ordonnées pour le prochain sprint.

7.1.2. Définition de « prêt » pour une story

C'est l'équipe qui décide si une story est prête. Pour prendre sa décision, elle pourra s'appuyer sur une liste de vérifications, qu'on appelle la *définition de prêt*.

Vérifier les « 6D »

La définition de prêt est élaborée par l'équipe et dépend du contexte. Une façon simple pour démarrer est de vérifier si la story est « 6D » :

- **Décomposée** : ce n'est plus une story épique.
- **Débattue** : elle a été discutée en équipe lors des séances d'affinage, au cours de conversations.
- « **Dérisquée** » : néologisme pour signifier que les risques sur cette story sont réduits.
- Elle possède une **définition de fini** : pour qu'elle soit prête, c'est bien de savoir ce qui permettra de dire qu'elle est finie.
- **Démontrable** : on saura la montrer à la revue, lors de la démo.
- **Désirable** : pour être sûr que la story apporte de la valeur à quelqu'un.

Exemple de vérifications

La story « Ajouter un parcours de promenade de chien » vérifie les « 6D » (tableau 7.1).

Tab. 7.1

<input checked="" type="checkbox"/>	Décomposée	Ce n'est plus une épique, sa taille est de 2 points.
<input checked="" type="checkbox"/>	Débattue	Le 18 juin, lors de la séance d'affinage.
<input checked="" type="checkbox"/>	Dérisquée	Laurent, l'expert de la cartographie, est disponible 3 jours.
<input checked="" type="checkbox"/>	Définition de fini	C'est une user story, qui reprend les vérifications de fini pour ce type de story.
<input checked="" type="checkbox"/>	Démontrable	La condition d'acceptation positive d'ajout de parcours après la promenade est décrite.
<input checked="" type="checkbox"/>	Désirable	Kevin, le directeur commercial, y tient beaucoup pour ses clients urbains.

Les vérifications sont examinées lors des conversations d'affinage. Certaines font référence à une information associée à la story, qui sera étudiée de façon qualitative.

7.1.3. Prêt pour une feature

Commencer le développement d'une feature non prête présente aussi des risques. Se poser la question de savoir si elle est prête permet d'éviter ce que j'ai rencontré un jour : les 30 features d'une release étaient commencées, mais aucune n'était finie à moins de deux sprints de la fin.

Les risques sont souvent liés à des dépendances : à des experts fonctionnels, à des composants, etc.

7.2. L'affinage, une pratique d'équipe

L'affinage du backlog est une activité faite par l'équipe pendant un sprint, dans le but de préparer des stories pour les sprints suivants.

7.2.1. Avec qui ?

Bien évidemment, le Product Owner est le principal affineur. Il affine parfois seul, mais le plus souvent avec l'équipe. Il est souhaitable que toute l'équipe Scrum participe à cette activité.

Les experts sont invités selon le besoin. Des experts fonctionnels sont utiles quand leur domaine est abordé.

7.2.2. Dans quel cadre est-il pratiqué ?

Une grande partie de ce travail est constituée de conversations entre le Product Owner et le reste de l'équipe.

On donne, de préférence, un caractère permanent, officiel, à l'affinage en le plaçant dans le cadre d'une réunion d'équipe : la réunion d'affinage de backlog.

Cela permet de considérer que l'affinage fait partie du travail lié à Scrum, comme les événements du sprint. Ainsi, il n'est pas nécessaire de faire figurer la plupart des travaux d'affinage explicitement dans le backlog.

7.2.3. Quand ?

Il y a deux possibilités pour l'organisation des réunions d'affinage :

- **À un rythme régulier** – Par exemple, tous les mercredis. C'est le plus facile, en particulier quand on débute.
- **Sur demande** – C'est plus agile : le ScrumMaster surveille les stories prêtes et s'il constate qu'il y a un risque de disette, il déclenche un affinage de backlog (voir le chapitre 20 *Appliquer Kanban sur Scrum*).

Une équipe qui consacre entre 1 h 30 et 2 heures par semaine à l'affinage, cela fait environ 5 % de son temps et c'est raisonnable. Le guide Scrum 2013, qui ne donne pas de minimum, évoque un maximum de 10 %.

7.2.4. Avec quoi ?

Un backlog structuré avec des bacs facilite le travail d'affinage, mais ce n'est pas indispensable : le bac n'est que le reflet de l'état de la story.

- L'affinage ayant pour objectif de produire à temps des stories prêtes, le **bac de départ** permet de bien visualiser la réussite de cet objectif : c'est facile de savoir s'il en contient assez.
- Les autres stories ont naturellement leur place dans le **bac d'affinage**.
- Les idées, les demandes, les feedbacks pour lesquels le Product Owner n'a pas encore statué vont dans le **bac à sable**.

Nous avons donc un bac où on va affiner et deux autres bacs qui correspondent à des files d'attente, avant et après.

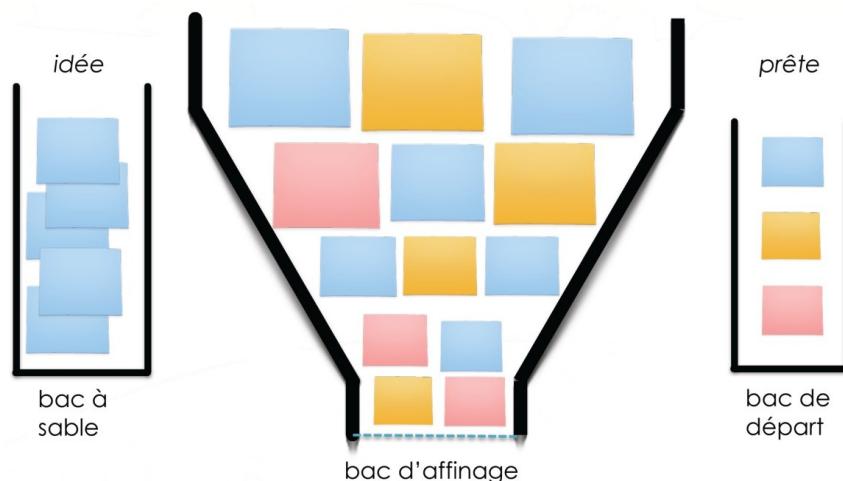


Fig. 7.1 Affinage, de l'idée au départ du sprint

L'idée des bacs m'est venue alors que j'étais Product Owner d'iceScrum, en 2010. Il y a eu le bac à sable, puis s'est ajouté un bac à glace. Pour iceScrum, ça s'est arrêté là, mais j'ai continué à expérimenter cette façon de présenter le backlog. C'est devenu « les bacs ». L'expérimentation a été un succès. J'ai présenté « les bacs » dans la troisième édition de mon livre. J'en ai parlé dans les conférences. J'ai poussé des équipes à les utiliser. D'autres ont essayé et ont trouvé ça bien^[1].

Cette façon de structurer le backlog contribue à mieux comprendre le cycle de vie de la story avant le sprint. Elle s'appuie sur l'intérêt croissant pour le management visuel et sur la diffusion de Kanban[Morisseau, *Kanban*, p.245]

7.3. Les activités d'affinage

Le détail et l'ordre des activités menées lors d'une séance d'affinage dépendent de la situation. Voici une séquence possible (figure 7.2) :

1. On regarde le bac de départ. S'il ne contient pas assez d'éléments, **l'approvisionner en stories prêtes** est primordial.
2. On observe ensuite le bac d'affinage pour identifier les stories épiques qu'il faut **décomposer**.
3. On examine le bac à sable et le tableau de features en vue d'**approvisionner en stories à affiner**.
4. On **estime** les éléments approvisionnés ou décomposés qu'il contient.

5. On réordonne le bac d'affinage.
6. On purge les bacs.

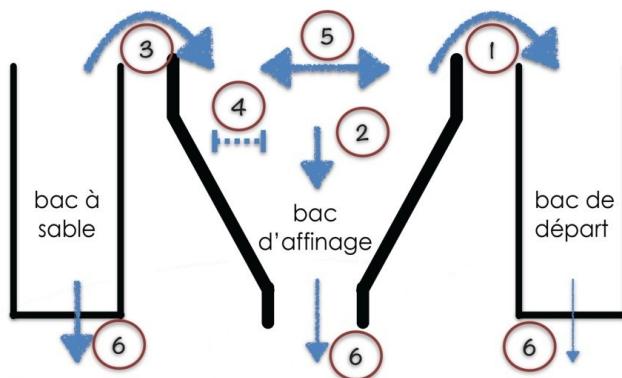


Fig. 7.2 Les activités d'affinage

7.3.1. Approvisionner en stories prêtes

La première chose à faire est d'évaluer le besoin en nouvelles stories prêtes.

Pour une équipe qui fait en moyenne n stories par sprint, il convient d'avoir, vers la fin du sprint, entre $1,2n$ et $1,5n$ de stories prêtes pour le suivant.

Ensuite, l'équipe examine les stories les plus prioritaires du bac d'affinage. La conversation avec le PO a pour objectif de compléter leur connaissance. La confirmation qu'une story est prête est obtenue quand l'équipe estime que les risques de ne pas la finir dans le sprint sont réduits. Elle s'appuie sur la liste des vérifications de la définition de prêt, ou sur les « 6D ».

On continue ainsi, story par story, jusqu'à en avoir assez de prêtes.

Cette activité est faite en premier : dans le cas où il manque beaucoup de stories prêtes pour le sprint suivant, on y passera le temps qu'il faut.

Les discussions sur les risques peuvent prendre un caractère technique. Souvent, l'équipe réfléchit à la conception de la story pour réduire le risque sur sa réalisation. Certaines équipes identifient précocement les tâches, petits morceaux de travail, et se mettent d'accord sur le référentiel de la story. Pourquoi pas si cela contribue à rendre la story prête. Parfois, la discussion technique dure, le Product Owner en profitera alors pour préparer l'activité suivante, décomposer.

7.3.2. Décomposer

Une fois le bac d'affinage débarrassé des stories prêtes, les stories épiques qu'il faut décomposer en priorité apparaissent : ce sont celles qui sont susceptibles d'alimenter le prochain sprint ou le suivant. Pour les autres, on a le temps.

Les épiques sont décomposées en stories plus petites. Les techniques de décomposition sont approfondies dans le chapitre 15 *Raconter la story*.

Tout l'art de l'affinage est de décomposer au bon moment : le faire trop tôt produit du stock, le faire trop tard risque de mettre en péril le prochain sprint. Une fois la décomposition terminée, l'épique disparaît, il ne reste que les stories.

On pourrait décomposer le bac d'affinage en deux sous-bacs, un pour les épiques et l'autre pour les stories. Mais comme il arrive qu'une épique puisse être plus prioritaire qu'une story, il vaut mieux ne conserver qu'un seul bac et utiliser un signe visuel pour les différencier. Cette distinction est faite par couleur, forme de Post-it®, gommette, indication ou toute autre idée qui convient à l'équipe.

On peut changer d'avis : une story peut redevenir épique. Savoir si la décomposition est bonne demande de l'expérience. Une façon de le savoir, c'est d'estimer la taille des stories.

7.3.3. Approvisionner en nouvelles stories

Le bac d'affinage est alimenté à partir du bac à sable ou depuis le tableau de features.

Depuis le bac à sable, le PO et l'équipe discutent pour décider quoi faire des propositions. Les choix qui s'offrent :

1. passer la story dans le bac d'affinage dans le but de l'inclure dans la release courante ;
2. la supprimer si elle n'apporte pas de valeur ;
3. considérer que c'est un morceau beaucoup plus gros qu'une story même épique en la transférant dans le tableau de features.

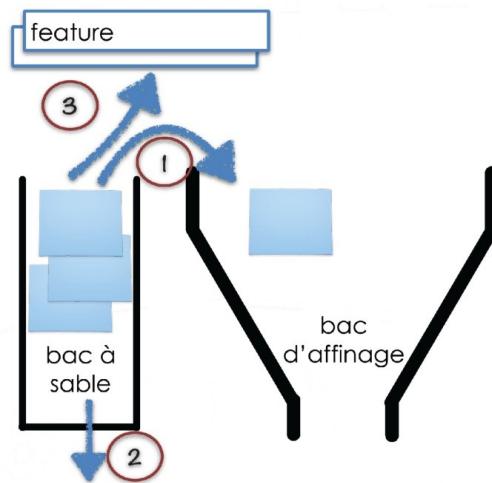


Fig. 7.3 Sortie du bac à sable

Depuis le tableau de features, la décision d'en commencer une nouvelle conduit à une discussion, pour la décomposer grossièrement en épiques.

Un point à examiner est l'avancement des features : il est recommandé d'en finir une plutôt que d'en commencer une nouvelle.

Des stories peuvent être ajoutées de façon automatique au cours de l'affinage, sans passer par le bac à sable : ce sont les stories de finition de features, de finition de sprint et les stories de réserve pour urgences. Nous reviendrons sur l'utilité de ces notions avancées dans le prochain chapitre.

7.3.4. Estimer

Cette activité porte sur les stories ajoutées ou celles décomposées depuis la dernière séance d'affinage. Les estimations sont utiles pour qualifier une story comme « trop grosse », qui ne peut donc pas passer prête.



Fig. 7.4 Cette story est estimée trop grosse

Elles servent aussi à faire des prévisions pour la release, si on en a besoin.

Elle se pratique souvent avec du *planning poker*. Ce « jeu » est un prétexte à des conversations entre le PO et l'équipe, utile lors les premiers sprints, quand l'équipe ne maîtrise pas bien les activités d'affinage.

Nous verrons les autres enjeux et les pratiques de l'estimation dans les chapitres 16 *Planifier la release* et 18 *Améliorer la visibilité avec des indicateurs*.

7.3.5. Revoir l'ordre

En même temps qu'on ajoute des éléments au bac d'affinage, on les ordonne rapidement. Ils ne se placent pas systématiquement à la fin, après ceux déjà présents.

Le Product Owner aura réfléchi à l'ordre et le présente à l'équipe pour discussion. La conversation qui s'ensuit porte notamment sur la valeur relative des stories et surtout sur les dépendances.

Les développeurs vont parfois défendre la priorité d'une story technique qui leur paraît importante. Toute l'équipe participe à la définition des priorités, mais en cas de désaccord,

c'est le PO qui arbitre.

Un bac d'affinage contient tout au plus deux ou trois dizaines d'éléments et l'ordre est simplement actualisé à chaque réunion, ce qui explique que cette activité puisse se faire rapidement.

L'ordre (ou priorité ordinale) correspond au rang définissant la séquence de réalisation des stories. L'équipe s'appuie sur les critères **valeur, taille et dépendances** pour définir l'ordre.

- La valeur est la somme de la « valeur métier » et de la « valeur de connaissance », ce que la story permet d'apprendre, permettant de réduire un risque fonctionnel ou technique.
- La taille est celle qui est estimée dans l'activité précédente.
- Les dépendances sont de trois sortes, elles portent sur d'autres stories, sur les gens qui peuvent réaliser la story, ou sur une échéance temporelle.

7.3.6. Purger

Il convient de purger régulièrement le backlog. Les éléments qui y stagnent trop longtemps sont des bons candidats à la purge.

Éliminer des éléments du bac à sable est une activité récurrente du Product Owner. Parmi les demandes qui sont faites dans ce bac, c'est à lui de décider lesquelles il veut intégrer à son produit ou son service. Plutôt que de laisser traîner dans le bac à sable une demande qu'il ne veut pas satisfaire, il la supprime.

Enlever des éléments qui sont entrés dans le bac d'affinage est aussi possible. Le PO les a fait entrer comme des options, il n'y a pas d'engagement à ce niveau-là.

En outre, dans le bac d'affinage, on trouve des stories qui sont le résultat de l'activité de décomposition. Il est tout à fait possible que certaines soient jugées inutiles pour la release courante.

Éliminer des stories déjà prêtes qui sont dans le bac de départ doit rester exceptionnel, mais cela peut arriver. Il est préférable de rejeter à ce moment que de passer du temps à réaliser quelque chose qui n'apporte pas de valeur.

Éviter de faire du travail inutile est le premier bénéfice de l'agilité.

Bac à ne pas faire

Pour éviter d'avoir à purger des choses dont il ne veut pas, un Product Owner peut maintenir et publier une liste des choses déjà rejetées, pour éviter que les mêmes demandes reviennent plusieurs fois.

Bac à glace

L'usage de ce bac est optionnel. Il correspond à une purge provisoire du bac d'affinage. Il a de l'intérêt pour les équipes pratiquant la release cadencée sur une durée fixe. Si on livre

à chaque fin de release, il y a sûrement des stories qu'on va développer, mais pas tout de suite, ce sera dans la prochaine release. Gelons-les dans le bac à glace, ce serait dommage que les développeurs perdent du temps dessus maintenant.



Fig. 7.5 À la fin de release tendue, un bac à glace soulage.

Les stories y sont gelées jusqu'à la *release* suivante. Quand celle-ci démarre, elles ont vocation à être dégelées dans le bac d'affinage.

Il ne s'agit pas d'y mettre toutes les demandes qui ne sont pas acceptées par le PO. Le bac à glace n'est pas une poubelle. Les demandes que le PO juge inutiles sont impitoyablement supprimées (si finalement on change d'avis, elles reviendront bien vite dans le bac à sable) et seules celles dignes d'intérêt pour la prochaine release seront gelées dans le bac à glace. La période de glaciation est limitée à quelques mois, le bac devrait être presque vidé au début de chaque nouvelle release.

Ce bac permet de limiter la taille des deux autres en stockant des stories qui n'ont pas d'intérêt à court terme pour l'équipe, mais qu'on ne veut pas supprimer car elles ont une valeur pour le produit.

Le bac à glace sert à ajuster le périmètre pour la release.

7.4. Affinage pendant le sprint zéro

Le premier affinage a lieu pendant le sprint zéro, dont c'est l'activité majeure. Il est particulier, dans la mesure où le travail est beaucoup plus important cette première fois qu'en régime courant.

Il dure bien plus longtemps qu'un affinage ordinaire, éventuellement plusieurs journées, car il y a beaucoup de stories, et il peut inclure des ateliers comme l'impact mapping et le

story mapping (sur lesquels nous reviendrons dans le chapitre 14 *Découvrir le produit*).

Les parties prenantes sont conviées à participer, au moins à la première partie, qui a pour objectif d'alimenter le tableau de features.

Une fois les features identifiées et ordonnées, les activités présentées plus haut pour un affinage standard s'appliquent, généralement dans cet ordre : approvisionner le bac d'affinage, ordonner, décomposer, estimer et, pour finir, approvisionner le bac de départ pour le sprint 1.

Si l'équipe ne dispose d'aucune donnée historique sur le nombre de stories qu'elle peut traiter dans un sprint, il conviendra de prendre de la marge lors de l'approvisionnement du bac de départ.

7.5. Résultat de l'affinage

7.5.1. Anticipation avec un backlog affiné

Un backlog affiné est dans un état permettant de commencer le prochain sprint dans avec une bonne confiance. En particulier, l'affinage permet d'éviter une surcharge de travail lors de la planification du sprint, et contribue à un engagement librement consenti de l'équipe.

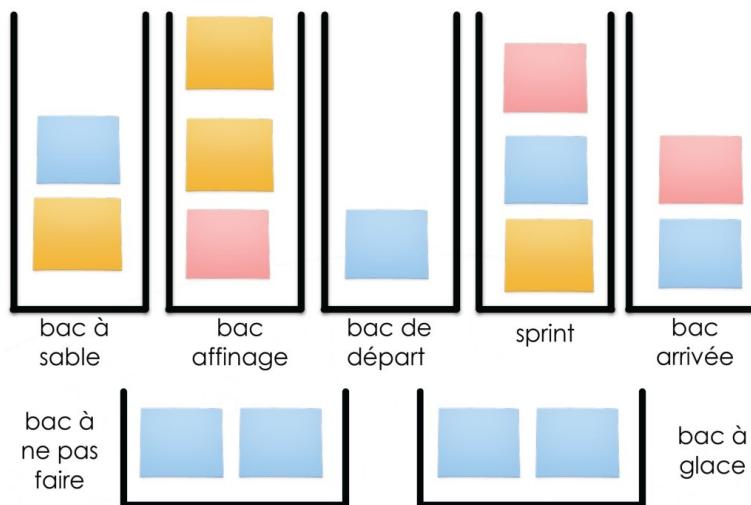


Fig. 7.6 Structure du backlog en 4 + 2 + 1

L'équipe a aussi une meilleure connaissance de ce qui est prévu dans les sprints futurs. L'élaboration d'un plan de release est grandement facilitée.

7.5.2. Story de découverte

Il arrive qu'il soit nécessaire d'analyser une demande mise dans le bac à sable ou une story épique du bac d'affinage. La plupart du temps, cela se fait pendant l'affinage collectif.

Parfois, l'étude nécessite plus de temps et le PO n'est pas en capacité de la réaliser. Il est alors possible de déléguer le travail à l'équipe.

Pour être conforme au postulat que tout travail soit visible, il faudrait que ce travail figure dans le backlog. On peut donc créer une story, dite de découverte, dont l'objectif est de créer ou préciser des stories. Elle sera affinée comme les autres.

7.6. L'affinage sur le terrain

7.6.1. Attention à ne pas restreindre l'affinage au planning poker

Le planning poker est un atelier permettant d'estimer collectivement les stories. Sa vertu essentielle est de pousser à des conversations dans l'équipe et avec le Product Owner. Certaines équipes désignent sous l'appellation planning poker la réunion d'affinage. Il y a un risque de négliger l'essentiel, qui n'est pas l'estimation, mais la conversation pour rendre la story prête et pour pouvoir la finir en un sprint.

7.6.2. Éviter de stocker

L'affinage est une pratique émergente qui vise à avoir suffisamment de stories prêtes. Il serait dommage, par excès de zèle, d'avoir trop de stories prêtes à l'avance. Dans la même idée, décomposer trop tôt des stories épiques augmenterait le stock dans le bac d'affinage.

Nous verrons dans le chapitre 20 *Appliquer Kanban sur Scrum* des techniques permettant de limiter ce risque.

7.6.3. Faire deux réunions d'affinage par sprint

La réunion d'affinage a lieu au moins une fois pendant chaque sprint. Le nombre de séances dépend de la longueur du sprint.

Pour débuter, et même pour les sprints de deux semaines, je conseille d'en prévoir deux pour orienter la deuxième, si nécessaire, vers l'objectif principal, avoir suffisamment de stories prêtes pour le prochain sprint.

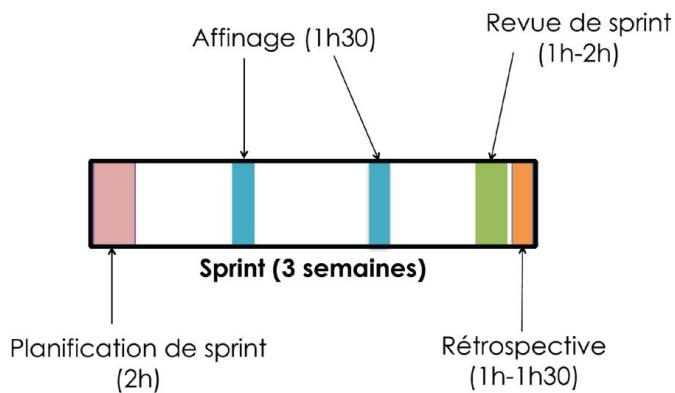


Fig. 7.7 Les réunions d'affinage pendant un sprint

7.6.4. Inviter des experts

Lorsqu'il existe des dépendances sur des compétences à l'extérieur de l'équipe, il est préférable de faire participer ces experts aux séances d'affinage. Cela permet d'avoir une conversation avec eux, de mieux les impliquer et ainsi de réduire le risque associé à cette dépendance.

Bien commencer

La question à se poser

Est-ce que nous embarquons dans le sprint des stories dont on se dit par la suite qu'on n'aurait pas dû les prendre ?

De mauvais signes

Les développeurs ne finissent pas les stories à cause de dépendances externes.

Il n'y a pas de définition de prêt.

Il n'y a pas de séance régulière d'affinage.

Par quoi démarrer

Un dépôt pour les stories prêtes afin qu'elles soient visibles.

Une lecture pour tous

La vidéo [Kniberg, Product Ownership] qui présente, entre autres, l'essentiel sur le backlog et son affinage dans une superbe animation de quinze minutes.

À retenir

L'affinage est une pratique qui permet de conserver un backlog prêt à l'emploi, avec des stories prêtes.

La définition de prêt permet d'affermir la confiance de l'équipe à finir une story dans un sprint. Elle se base sur des vérifications, par exemple les « 6D ».

L'affinage se pratique en équipe, lors des réunions d'affinage du backlog, avec des conversations sur les stories. Il a lieu pendant un sprint et influence le contenu des sprints suivants.

Il consiste à détailler, décomposer, ordonner, approvisionner, estimer et purger. Cela est facilité par une structure du backlog basée sur un tableau de features et sur les 4 + 2 + 1 bacs.

Références :

☞ Gojko Adzic & David Evans, *Fifty quick ideas to improve your User Stories* 2014.

<http://www.50quickideas.com/>

☞ Henrik Kniberg, Product Ownership, VF, 2013.
https://www.youtube.com/watch?v=vkYEqz_MA5Y

☞ Laurent Morisseau, *Kanban pour l'IT*, Dunod.

☞ Jeff Sutherland et Ken Schwaber, Scrum Guide, 2013, VF.

<http://wiki.ayeba.fr/Guide+Scrum+2013>

Notes

[1] Un retour d'expérience « *les bacs de culture c'est kiss* » a été publié par un expérimentateur rock'n roll (si on considère que Kiss c'est du rock... ce qui n'est pas mon cas). Consultez l'article sur <http://jfallet.wordpress.com/2014/01/02/les-bacs-de-culture-cest-kiss/>

La définition de fini

« *Est-ce que tu as fini ton travail ? Quand aurez-vous fini de coder ? Est-ce que la doc est finie ? Est-ce que tu as fini de ranger ta chambre ?* »

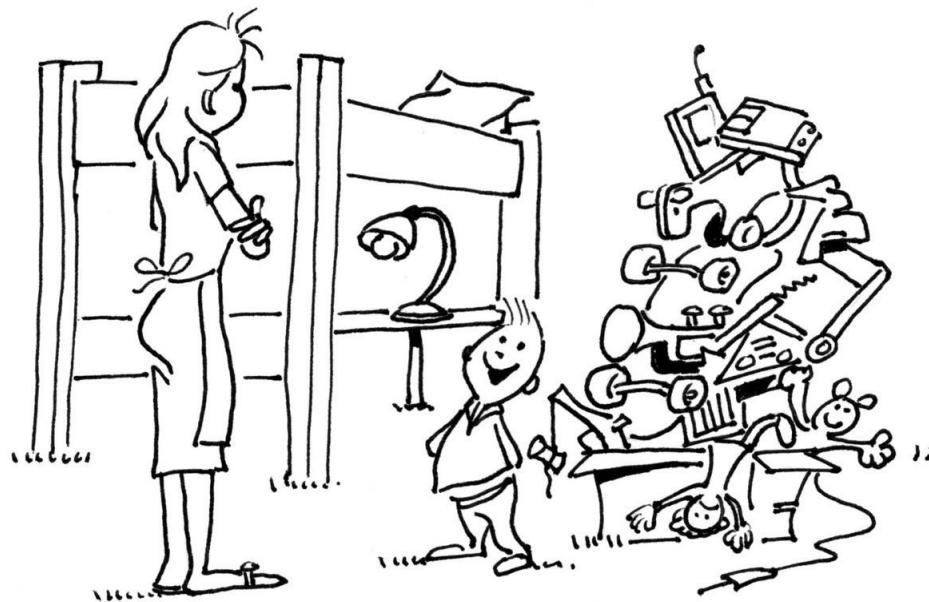


Fig. 8.1 « *Fini !* »

Ces questions reviennent régulièrement dans le développement d'un produit, comme dans la vie. Avec Scrum, c'est tellement important qu'il existe une pratique dédiée à cela.

À la fin d'un sprint, l'équipe obtient un résultat. Mais cela ne suffit pas : il faut qu'il soit « fini ». Qu'est-ce que cela veut dire ? La réponse varie, c'est à chaque équipe de le définir et de l'appliquer. Le constat fait avec les équipes qui démarrent en Scrum est que cette pratique est difficile à réussir : on pense que c'est fini, mais c'est seulement presque fini.

En général, la signification de fini n'est pas la même pour tout le monde.

L'objet de ce chapitre est de montrer comment procéder pour que « fini » ait la même signification pour toute l'équipe.

8.1. Finir l'incrément de produit

Le tempo dans Scrum est donné par le sprint, de durée fixe et courte. Si on n'y prend pas garde, le risque existe qu'à la fin du sprint, le résultat ne soit pas satisfaisant pour les parties prenantes ou pour les développeurs. Ce résultat, c'est l'incrément de produit.

C'est pour réduire le risque sur le contenu de cet incrément qu'on donne un **objectif au sprint**.

C'est pour réduire le risque sur la qualité de l'incrément qu'on a une **définition de fini**.

8.1.1. L'objectif du sprint

Chaque sprint possède un objectif, défini au début et vérifié à la fin. Il prend la forme d'une phrase résumant le contenu sur lequel l'équipe va se focaliser et s'engager devant les parties prenantes.

L'objectif du sprint est la pratique Scrum qui renforce les valeurs de focalisation et d'engagement de l'équipe. Il permet d'aligner toute l'équipe, comme le cap pour un navire.

Son usage est présenté dans le prochain chapitre.

8.1.2. La définition de fini

Pour que le sprint soit un succès, son objectif doit être atteint, mais il faut aussi que l'incrément soit dans l'état souhaité, décrit par la définition de fini.

La notion d'échec d'un sprint est à prendre au sens Scrum : il est mieux d'échouer le plus tôt possible, c'est une bonne façon d'apprendre et cela laisse du temps pour se réorganiser.

Le terme anglais est « *Definition of Done* », que certains francophones utilisent. De mon point de vue, fini ou terminé conviennent bien.

Comme l'incrément comprend des stories et des features, la définition de fini porte également sur l'état souhaité de ces éléments.

La définition de fini est la pratique Scrum à part entière qui garantit la qualité de l'incrément du sprint ; elle produit l'artefact éponyme qui consiste en une liste de vérifications sur la story, la feature et le sprint.

8.2. Finir les stories et les features

La métaphore du sprint n'est pas adaptée aux développeurs, mais elle garde son sens si on considère que ce sont les stories qui sprintent. Elles ne finissent pas toutes en même temps.

8.2.1. Visualiser la finition

L'incrément est obtenu en fin de sprint, en revanche les stories ou features qu'il contient se finissent pendant le sprint, au fil de l'eau.

Story

Pendant qu'on travaille à sa réalisation, une story est en course. Dès qu'elle est finie, elle passe dans le bac d'arrivée. On n'attend pas la fin du sprint pour l'officialiser.



Fig. 8.2 « This is the end. »

L'équipe Peetic avec une équipe de cinq développeurs réalise en général une dizaine de stories, ce qui fait, en fin de compte, pas loin d'une finie par jour.

Feature

Finir une story, c'est bien, finir une feature, c'est mieux !

Dans un tableau de features, les colonnes sont représentatives de l'état. Selon la finition de la feature, on la fera passer dans la colonne suivante.

L'équipe Peetic, dans une release de cinq sprints, a en général entre une demi-douzaine et une dizaine de features qui sont finies, ce qui fait une ou deux par sprint. En moyenne, car il peut y avoir des sprints, le premier en particulier, sans feature finie.

8.2.2. Définition de fini pour une story

Dans un vrai sprint, un coureur a fini quand il passe la ligne d'arrivée, mais on vérifie qu'il a bien son dossard et qu'il n'a pas mordu sur la ligne du voisin. Pour qu'une story soit finie, il faut qu'elle passe avec succès ses conditions d'acceptation, mais il y a aussi des vérifications annexes à effectuer.

La définition de fini pour une story est une liste de vérifications.

Bien que chaque story soit différente, des vérifications sont communes à plusieurs stories. On crée alors une liste valable pour plusieurs stories. Les stories qui partagent la même liste appartiennent au même *storyotype*, pour reprendre le terme lancé par Gerald Mesrazos et repris dans Exploring Scrum [Rawsthorne, *Exploring*].

Un *storyotype* correspond à un ensemble de stories partageant la même liste de vérifications de fini et de prêt.

Nous avons vu dans le chapitre 6 *Structurer le backlog* plusieurs types de stories. La notion de storyotype va plus loin. Dans le type story fonctionnelle, on trouvera le storyotype « user story contenant du code » ; cependant ses vérifications ne s’appliquent pas à toutes les stories fonctionnelles, en particulier celles n’incluent pas de code.

Pour l’instant, nous allons en rester au storyotype « user story » et, comme exemple, nous allons prendre la story : Ajouter une photo de son animal.

La première vérification porte sur la condition d’acceptation.

Notre story possède sa condition d’acceptation de succès : la photo apparaît en vignette sur la page animal.

La façon de décrire les conditions d’acceptation sera vue plus loin.

Si on veut éviter de produire de la dette technique, il convient d’effectuer d’autres vérifications. On peut les classer selon qu’elles portent sur des aspects visibles par les parties prenantes (utilisabilité, fiabilité, performance, sécurité, etc.) ou perceptibles uniquement par l’équipe (maintenabilité, réutilisabilité, etc.).

Pour Peetic, la définition de fini contient les vérifications « développeur » : codée en suivant le standard de codage, revue de code effectuée, tests unitaires passés.

Ainsi que des vérifications « partie prenante » : tests d’acceptation passés avec succès dans un environnement de « pré-prod », et version disponible en français et en anglais.

8.2.3. Fini pour une feature

Pour qu’une feature soit finie, les stories qui la composent doivent l’être, et, de plus, des vérifications supplémentaires sont nécessaires.

Par exemple, il faut s’assurer que l’enchaînement des stories est bon, par des tests de plus haut niveau. On va aussi y trouver des vérifications qu’il ne serait pas opportun d’effectuer sur une story.

Pour Peetic, une *feature* est finie si elle inclut la rédaction de la documentation marketing, la vérification de l’export en pdf, l’ajout de raccourcis clavier.

Pour ne pas oublier les travaux induits par les vérifications, je conseille de créer une story de finition pour chaque feature.

8.2.4. Vérifier la finition

Pendant le sprint, les vérifications sont effectuées pour s’assurer que la story est bien finie. Il ne s’agit pas de contrôles faits par une entité extérieure, c’est l’équipe qui s’en charge, dans l’état d’esprit de produire un résultat de qualité.

Qui vérifie la finition ?

Des équipes pratiquent la vérification croisée, c'est-à-dire que c'est une personne n'ayant pas participé aux travaux sur la story qui s'en charge. Si cela part d'une bonne idée, cela crée un temps de latence, en attendant que la personne soit disponible pour cette vérification.

D'autres considèrent que c'est normalement le Product Owner qui décide de ce qui est fini. Mais certains PO, manquant d'implication, ne sont pas assez disponibles pour vérifier les conditions d'acceptation. En outre, souvent à cause de leur expérience d'utilisateurs face à des informaticiens, ils sont méfiants, ont du mal à entériner que quelque chose est vraiment terminé. Avoir une définition élaborée et vérifiée par toute l'équipe évite d'avoir une trop forte dépendance vis-à-vis de la décision du Product Owner.

C'est l'équipe qui décide qui vérifie la finition. Les vérifications pour une story peuvent être nombreuses et partagées entre plusieurs personnes.

Que faire si une vérification ne passe pas ?

On fait en sorte qu'elle passe avant la fin du sprint. Si on n'y arrive pas, la story n'est pas finie.

8.3. Finir le sprint et la release

8.3.1. Fini pour le résultat du sprint

À la fin du sprint, le résultat est l'incrément de sprint, placé dans un environnement accessible aux parties prenantes.

Pour Peetic, il est taggé dans Git et déployé en pré-production.

Il contient des stories qui ont été vérifiées individuellement, mais d'autres vérifications sont possibles. La première porte sur l'objectif du sprint et c'est binaire : atteint ou pas.

Selon le contexte, les vérifications varient énormément. Pour certains projets, où la criticité est forte, les besoins en documentation seront importants, et la mise à jour des documents entre dans la définition de fini du sprint.

On y retrouve des vérifications sur le code qu'on ne fait pas, au moins au début, pour chaque story, mais globalement, par exemple :

- sur la couverture de tests,
- sur le respect du nombre de lignes pour une méthode de classe.

Des outils qui mesurent la qualité du code^[1] permettent d'obtenir des mesures utiles pour vérifier les conditions de finition. Si l'équipe possède un standard de codage, la définition de fini s'y réfère plutôt que reprendre chaque règle.

La définition de fini pour un sprint implique des travaux qui peuvent faire l'objet d'une story spéciale, appelée story de finition du sprint.

Même si les vérifications de fini pour un sprint n'ont pas été passées avec succès, le sprint se termine à la date prévue. Il faudra en tenir compte en ajoutant immédiatement une story pour rembourser la dette technique, et c'est un point à aborder lors de la rétrospective.

8.3.2. Fini pour le résultat de la release

Cela dépend de ce qu'on fait du produit à la fin de la période de temps que constitue la release. Est-il mis en production ? Ou bien est-il placé dans un environnement d'intégration ou de pré-production ? Des activités de marketing doivent-elles être réalisées ?

Quelle que soit la réponse, il est utile de définir l'état du produit pour le jalon que constitue la fin de release. Cela permet de se rendre compte de ce qu'il faudra faire à la fin si ce n'est pas pris en charge au fur et à mesure, dans les sprints. L'idéal est de tout faire pendant les sprints pour raccourcir au maximum cette période avant la release. Cela peut nécessiter d'ajuster la définition de fini à mesure que les sprints avancent.

8.4. Les activités pour définir fini et prêt

Nous avons étudié dans le chapitre précédent la petite sœur de la définition de fini, la définition de prêt. Nous avons vu que pour augmenter ses chances d'être finie, une story devait être prête au début du sprint. Les activités décrites traitent les deux définitions.

8.4.1. Établir la définition de fini et de prêt

Toute l'équipe est impliquée : c'est normal, c'est elle qui appliquera ces pratiques. La participation du Product Owner au travail collectif est indispensable et celle des parties prenantes est souhaitable.

Les définitions de fini et de prêt sont élaborées pour la première fois pendant le sprint zéro.

Ce travail ne fait pas l'objet d'une réunion définie dans Scrum, la façon de procéder non plus. Le plus important, c'est que ce travail se fasse en équipe. Un *brainstorming* est la formule la plus adaptée pour collecter les idées de chacun afin de construire une définition de fini et de prêt complète et partagée.

La réunion est animée par le ScrumMaster, qui mène la consolidation à partir des idées collectées. En particulier, il doit s'assurer que la voix des personnes les plus impliquées dans le test et la qualité soit entendue.

Je conseille de lister d'abord toutes les vérifications sur des Post-it®, puis de les associer, dans un deuxième temps, à un des quatre niveaux (story, feature, sprint, release). On dessinera des cercles concentriques, un par niveau, c'est plus facile pour visualiser l'ensemble et changer de niveau en cours de discussion. On fait l'exercice pour fini, puis pour prêt (figure 8.3).

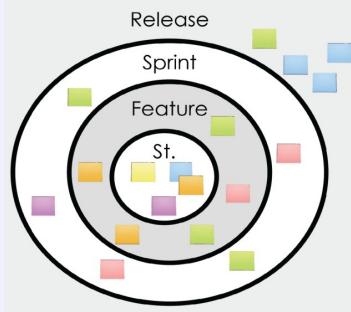


Fig. 8.3 Définition de fini sur les quatre niveaux

Les réflexions devraient notamment porter sur le niveau où placer les exigences non fonctionnelles ou transverses.

Pour le site Peetic : compatibilité avec Firefox 38.2, fonctionne sur tablette, aide en ligne disponible sous forme d'info-bulles sont placées au niveau feature.

L'objectif de cette première définition est d'obtenir une liste de vérifications qui couvrent au moins les user stories. Cette liste sera remise à jour à chaque sprint et complétée avec les autres types de story.

8.4.2. Publier la définition de fini et de prêt

La liste pour « prêt » et pour « fini » est publiée pour être bien visible de toute l'équipe. Il faut un moyen simple de communiquer, qui permet à tout le monde d'avoir une compréhension commune.

Quand l'équipe dispose d'un espace de travail ouvert, la publication prend la forme d'une affiche collée sur le mur à côté du tableau. On peut reprendre le résultat de l'atelier avec les vérifications sur des Post-it® placés dans les cercles correspondant aux niveaux.

La liste a également sa place dans les autres formes de communication utilisées par l'équipe : site, wiki, outil, etc. Dans ce cas, il est essentiel qu'elle soit très visible et apparaisse dans le tableau de bord.

Il ne s'agit pas d'avoir une liste trop longue qui serait difficile à suivre. On n'ajoute pas une vérification pour quelque chose d'évident pour toute l'équipe, du genre « code écrit ».

8.4.3. Évaluer et améliorer

La rétrospective est le moment où l'équipe s'interroge sur sa façon de travailler. Comme la définition de fini a un impact sur de nombreuses pratiques, cette réunion constitue le moment idéal pour examiner si elle a été respectée et, si nécessaire, l'améliorer.

C'est aussi au cours de la rétrospective que seront examinées en commun les raisons pour lesquelles une vérification n'est pas passée sur une story ou sur le sprint.

Les définitions de fini et de prêt sont susceptibles d'évoluer à chaque sprint, pour plusieurs raisons :

- Des aspects avaient été oubliés et ont été mis en évidence lors du sprint. D'autres sont devenus des habitudes, on les supprime.
- À travers la rétrospective, les pratiques d'une équipe évoluent régulièrement, à chaque sprint. La définition de fini et de prêt évolue également pour prendre en compte ces améliorations.
- On s'y interrogera aussi sur l'intérêt d'ajouter de nouveaux storyotypes.

Quand on se rapproche de la livraison, la qualité doit être ajustée. On augmente le niveau requis pour se rapprocher de la cible.

8.5. La définition de fini sur le terrain

8.5.1. Impact du mal fini

Ne pas bien mettre en œuvre la définition de fini pour une story a des impacts néfastes sur les sprints à venir. Les deux impacts les plus représentatifs sont la découverte de bugs, qu'il faudra corriger, et la production d'une dette technique, qu'il faudra rembourser avant qu'il n'y ait trop intérêts.

Dette technique

Ward Cunningham est l'inventeur du terme *technical debt*^[2]. Il utilise l'analogie avec la dette financière pour mettre en évidence que, comme les intérêts s'accumulent progressivement lorsqu'on contracte une dette, le coût de développement augmente si le logiciel comporte des imperfections techniques.

Chaque minute supplémentaire passée sur du code de mauvaise qualité correspond à l'intérêt de la dette.

La dette technique n'est perçue que par les développeurs (et encore pas toujours), les autres parties prenantes n'y sont pas sensibilisées. Comme son nom l'indique, elle est technique et cela peut avoir de multiples formes : architecture bancale, code mal écrit, standard de codage mal suivi, absence de documentation ou manque d'automatisation des tests.

Les mesures de prévention sont du ressort de la définition de fini, tandis que celles de curation constituent des stories de type remboursement de dette technique.

Bug

Une définition de fini incomplète ou mal appliquée explique la découverte ultérieure de bugs.

L'équipe Peetic souhaite que chaque user story finie possède une aide en ligne en anglais. On développe une user story d'inscription et, à la fin du sprint, comme ses tests d'acceptation sont passés avec succès, on la déclare finie.

Plus tard, dans un autre sprint, on s'aperçoit qu'il n'y a pas de texte en anglais dans l'aide. C'est un bug qui est ajouté dans le backlog.

8.5.2. Impact du pas fini

Perte de confiance

Si l'objectif du sprint n'est pas atteint ou si des vérifications ne passent pas, leur mise en évidence permet à l'équipe de s'améliorer pour le prochain sprint. En revanche, si le constat perdure sprint après sprint, les parties prenantes perdront leur confiance dans l'équipe, voyant qu'elle ne tient pas ses engagements.

Souvent, les équipes mettent la barre trop haut en début de projet, ce qui leur renvoie une image négative. Il est préférable de commencer avec ce qu'on sait tenir et de progresser régulièrement.

Report à la fin

Tout ce qui n'est pas fini pendant les sprints, et qui doit cependant l'être pour que les utilisateurs puissent utiliser le produit, constitue du travail qui devra être fait avant de livrer.

S'il est conséquent, il y a le risque de devoir y travailler en urgence pendant le dernier sprint.

8.5.3. Faciliter la vérification avec les cases à cocher

Les conditions à vérifier pour qu'une story soit finie constituent une liste associée à une story. Le suivi des vérifications est facilité par l'utilisation de cases à cocher.

Une variante est d'utiliser une matrice avec en lignes la liste des vérifications possibles sur les stories et en colonnes les stories sélectionnées pour le sprint courant.

Pour chaque story, une croix dans une ligne signifie que le contrôle correspondant est à faire. La matrice est définie par l'équipe en début de sprint.

8.5.4. Capitaliser avec la notion de storyotype

Pour Peetic, voici la description du storyotype « user story ».

User story	Apporte de la valeur à une ou plusieurs parties prenantes qui utilisent le produit
Définition de prêt	On a repris les « 6D » : décomposée, débattue, dérisquée, avec une définition de fini, démontrable, désirable.
	Codée en suivant le standard de codage, revue de code effectuée, tests unitaires passés,
Définition de fini	

tests d’acceptation passés avec succès dans un environnement de « staging »,
version disponible en français et en anglais,
documentation utilisateur rédigée.

Chaque équipe possède ses propres storyotypes, car ils dépendent du contexte.

Voici quelques exemples pour Peetic, en plus de « user story » :

- Story de support « hot line ».
- Story de découverte.
- Story de finition de feature.
- Story de finition de sprint.
- Étude technique (spike).
- Story d’infrastructure.
- Story de formation.

Bien commencer

La question à se poser

Est-ce que nous accumulons de plus en plus de dette technique ?

De mauvais signes

Pas de définition de fini publiée.

La définition de fini existe, mais elle n'est pas appliquée strictement.

Par quoi démarrer

Une équipe qui démarre s’appliquera d’abord à définir un objectif de sprint raisonnable. C'est la clé pour bien appliquer la définition de fini.

Une lecture pour tous

La procrastination [Perry, *Procrastination*], pour sa présentation de l'art de reporter au lendemain, et aussi du plaisir de finir un travail.

À retenir

La définition de fini est la pratique qui permet d’obtenir le niveau de qualité attendu pour l’incrément de produit obtenu à la fin d’un sprint, et pour éviter d’accumuler de la dette technique.

Elle se définit sur plusieurs niveaux : fini pour une story, fini pour une feature, fini pour un sprint, fini pour une release.

Les définitions de fini et de prêt évoluent constamment. Elles sont le reflet du niveau de maîtrise de l'équipe.

Références :

- ☞ John Perry, *La procrastination*, Marabout Pocket, 2014.
- ☞ Dan Rawsthorne & Doug Shimp, *Exploring Scrum*, 2012.

<http://agilelib.net/?100002>

Notes

- [1] Comme Sonar, outil Open Source : <http://sonar.codehaus.org/>
- [2] Dans un article publié en 1992 : <http://c2.com/doc/oopsla92.html>

Planifier le sprint

Dans le domaine des prévisions en informatique, de nombreux exemples illustrent les difficultés rencontrées. En voici trois qui résument des situations typiques dans des organisations où la gestion de projet est « traditionnelle » :

- Un développeur, investi dans son travail sur le code, n'aime pas trop qu'on lui demande quand il aura fini. Parfois, en insistant plusieurs jours, le chef de projet peut obtenir, après un ronchonnement, « *bon, je vais essayer de terminer demain* ». Cette attitude s'explique par l'habitude de ce chef de prendre une estimation pour un engagement et de mettre la pression après.
- On rencontre des chefs de projet qui font la planification tout seuls, identifient des grandes tâches, les « *chiffrent* » et les affectent aux personnes de leur équipe. Si les tâches n'avancent pas comme prévu, le chef de projet aura beau râler, les équipiers diront que les estimations n'étaient pas réalistes.
- Dans des organisations où le fossé s'est creusé entre l'IT et les autres métiers, le manque de confiance est tel que plus personne ne croit les dates annoncées par le chef de projet informatique. D'ailleurs les développeurs n'y croient pas non plus.

Pas de miracle avec Scrum pour mieux prévoir un avenir incertain ! Mais la planification du sprint propose une nouvelle façon de faire, qui s'appuie sur les valeurs de focalisation et d'engagement. Pour ramener de la confiance, on s'appuie sur le constat qu'on ne peut pas prévoir et s'engager de façon précise au-delà d'un certain horizon. L'horizon pour la planification détaillée correspond au sprint.

Cette pratique met aussi en évidence le rôle essentiel de l'équipe dans l'élaboration des plans. Ce n'est pas un chef qui définit ce qu'il y a à faire : le travail du sprint appartient à l'équipe, qui décide elle-même comment s'organiser.

Au-delà de sa fonction première de planification, la réunion est un rituel qui prépare l'équipe à travailler de façon collective pendant le sprint, comme les préparatifs dans les vestiaires amènent une équipe de rugby à rentrer dans son match avec un objectif partagé.

9.1. Les activités de planification du sprint

L'objectif de la planification est de mettre l'équipe en situation de réussir le sprint en se focalisant sur un objectif et s'accordant sur des stories.

9.1.1. C'est la story qui sprinte

Pour réussir son sprint, la story doit être bien placée, dans les « starting blocks », ce qui signifie qu'elle est prête.

Rappelons qu'une story est prête si elle est suffisamment petite et bien comprise par l'équipe, qui se sent alors capable de la finir sans risque dans un sprint.

L'affinage effectué pendant le sprint précédent a, en principe, permis d'obtenir suffisamment de stories prêtes.

9.1.2. C'est l'équipe qui planifie

Le résultat tangible de cette réunion est un plan avec les stories et les tâches associées. Mais le plan n'est pas l'essentiel, ce qui compte c'est la **planification** : les réflexions collectives faites au cours de ce rituel soudent l'équipe vers l'objectif du sprint.

Toute l'équipe participe à la planification.

Cela inclut le Product Owner. Comme la réunion est parfois longue et peut comporter des discussions techniques, la tentation est forte pour un PO de n'assister qu'au début, et de s'éclipser quand commence l'identification des tâches. Cependant, il est indispensable qu'il reste pour répondre aux questions que l'équipe va inévitablement poser. Il est obligatoire qu'il soit présent lors de l'engagement de l'équipe.

Les parties prenantes ne sont pas invitées, sauf les experts dont l'intervention est prévue pendant le sprint. Cela a été anticipé lors de l'affinage.

9.1.3. La durée de la réunion est limitée

Cette réunion est la première activité du sprint qui commence. Elle est limitée dans le temps, comme tous les événements du sprint.

Dans certaines publications sur Scrum, la planification du sprint est présentée en deux parties distinctes : la première pour définir le périmètre et le but du sprint, la seconde consacrée à l'identification des tâches nécessaires et à leur estimation. Historiquement, chacune de ces parties pouvait durer une demi-journée. Cela a bien changé : l'usage actuel, avec des sprints de deux ou trois semaines et la pratique de l'affinage du backlog, permet de réduire cette durée et de la faire d'un trait.

D'après mon expérience, on peut la mener en deux heures, même pour des sprints de trois semaines, et cela permet de maintenir la motivation des participants pendant la réunion.

9.1.4. Les tâches à faire apparaissent sur le tableau

Planifier le sprint passe par une décomposition du travail nécessaire pour réaliser chaque story. Les morceaux sont appelés des tâches.

Une tâche est un travail contribuant à une story et n'apportant pas de valeur par lui-même.

Ce sont les tâches qu'on voit, avec les stories, dans le tableau de sprint.

Un tableau de sprint Scrum sert à montrer l'avancement des travaux, c'est une représentation de son plan. Il est élaboré pendant la réunion. Pour chaque story sélectionnée, l'équipe identifie les tâches correspondantes. L'état des tâches est reconnu selon leur place dans les colonnes.



Fig. 9.1 Un tableau physique est hautement recommandé

Une équipe qui utilise les bacs de stories et le tableau de features visualisera les trois niveaux côté à côté.

Et si toute l'équipe n'est pas dans le même bureau ? Dans ce cas, on prendra des photos pour ceux qui sont loin ou on utilisera un outil informatique pour collecter les résultats.

9.2. Activités de planification du sprint

Voici un enchaînement possible des activités de la réunion.

- L'équipe se met dans le **contexte du sprint**.
- Pour chaque story prête, en commençant par la première, l'équipe **confirme avec le PO sa confiance** pour la finir dans le sprint.
- Pour faciliter la réalisation de la story, on prépare l'**essaimage** et on procède à l'**identification des tâches**.
- Avec cette connaissance du travail à faire, l'équipe s'**engage** sur l'objectif du sprint.
- Le sprint est alors **lancé** et chacun part réaliser une tâche.

9.2.1. Se mettre dans le contexte du sprint

Le Product Owner rappelle la place de ce sprint dans la release en cours.

Exemple : nous sommes au sprint 3 de la release « Vignemale » qui se terminera dans deux sprints.

Il donne la liste des stories prévues, en se basant sur l'équivalent de ce qui a été fait dans les sprints passés. Ce qui compte, c'est l'ordre de grandeur pour avoir une idée du nombre de stories au départ du sprint, pas pour un engagement précis.

Si la revue et la rétrospective du sprint précédent ont donné au PO des idées de stories qui lui paraissent intéressantes pour le sprint qui commence, il demandera à l'équipe de les affiner pour les ajouter à la liste du bac de départ.

9.2.2. Confirmer les stories prêtes avec le PO

Les stories ont déjà été discutées lors de l'affinage. L'objectif est maintenant de confirmer qu'elles peuvent prendre le départ du sprint. Pour chacune, on va donc vérifier la capacité de l'équipe à la réaliser pendant le sprint.

L'équipe Peetic a vérifié la disponibilité de Laurent, l'expert en cartographie, nécessaire pour réaliser la story « Ajouter un parcours de promenade de chien ».

En principe, comme les vérifications ont déjà été faites au moment de la déclarer prête, l'entente sur une story devrait se faire en quelques minutes.

Cependant une surprise de dernière minute peut être découverte, et l'équipe a le droit de reporter une story si elle estime que les conditions ne sont pas réunies. C'est mieux que de passer la moitié de la réunion dessus.

Il convient d'avoir un peu plus de stories que la moyenne habituelle d'un sprint, environ 20 % de plus.

L'accord conclu sur la story qui entre dans le sprint se base sur la liste des vérifications de la définition de prêt. Une story confirmée par cet accord rentre dans le sprint. L'entrée dans le sprint n'est pas encore définitive : les activités suivantes, orientées sur la réalisation et sur l'engagement, peuvent remettre en question le départ d'une story.

À l'issue de cette activité, on dispose d'une liste ordonnée de stories, qui peuvent déjà être visualisées dans la première colonne du tableau Scrum.

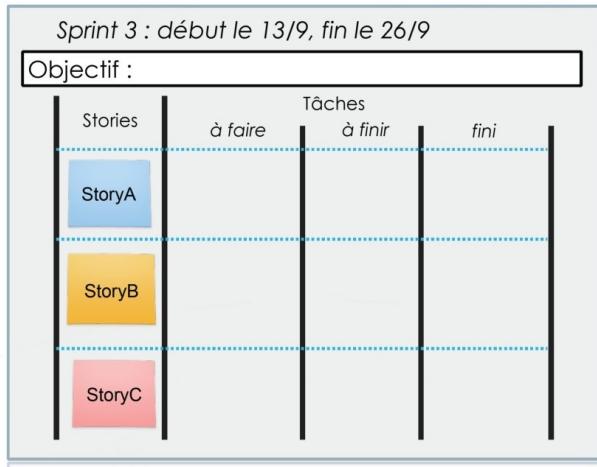


Fig. 9.2 Les premières stories dans le tableau

9.2.3. Préparer l'essaimage

La technique de l'essaimage favorise l'auto-organisation. En outre, elle vise explicitement à limiter le « multitâches », le « multistories » plus exactement, qui freine le déroulement du sprint, à cause du délai d'immersion.

Le délai d'immersion, provoqué par le fait d'arrêter un travail en cours pour en commencer un autre, est le temps nécessaire pour se remettre dans le bain du nouveau contexte.

Cela arrive quand un développeur ayant quitté une story doit y revenir pour modifier son code après le passage tardif d'un test.

L'essaimage consiste en une division temporaire de l'équipe. L'essaïm créé va s'agglutiner sur une story le temps de la finir.

À l'image de la colonie d'abeilles qui essaime avec une reine et des ouvrières, la story aura son référent et ses développeurs. À la différence d'un véritable essaim, la division d'équipe est temporaire.

Le référent de la story est un développeur qui connaît bien l'histoire. Il s'y est intéressé dès l'affinage. Il a pour mission de la raconter aux autres (quand le PO n'est pas là) et de faire en sorte qu'elle soit finie rapidement. Il reste attaché à la story jusqu'à ce qu'elle soit finie. Il ne peut être référent que pour une seule story en même temps.

L'essaimage consiste d'abord à définir le nombre de stories pouvant être traitées simultanément au sein de l'équipe. Selon la taille de l'équipe et le degré de spécialisation, cela peut varier.

L'idéal serait d'avoir un seul essaim à la fois, cela reviendrait à finir une story avant de passer à la suivante. Cependant pour la plupart des équipes, il paraît impossible de travailler à plus de trois ou quatre sur une story.

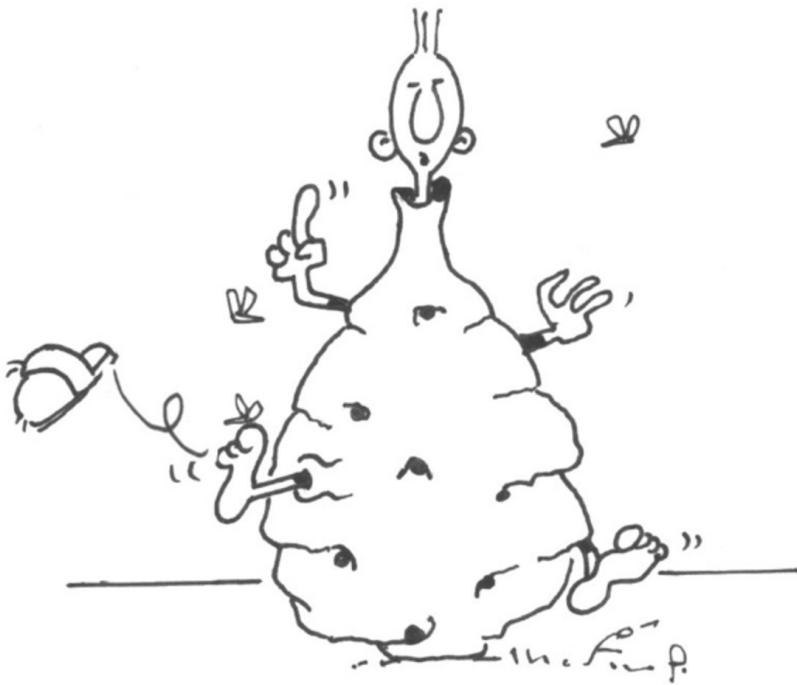


Fig. 9.3 Le référent raconte l'histoire à l'essaim.

Les essaims se créent par auto-organisation autour d'une story et décident eux-mêmes d'avoir un référent ou de s'en passer.

Une fois le premier tour de stories affecté, on passe à un deuxième et ainsi de suite. L'essaimage peut être anticipé lors de l'affinage. L'organisation des essaims évolue pendant le sprint, cela se décide lors des mêlées quotidiennes.

Les activités pour réaliser la story pendant le sprint, y compris la conception, seront faites au sein de l'essaim. Lorsque la story est finie, l'essaim n'existe plus, pour éviter de créer durablement des sous-équipes à l'intérieur de l'équipe Scrum.

9.2.4. Identifier les tâches

L'important est de finir des stories, les tâches n'étant qu'un moyen d'y arriver, dont on pourrait se passer. Cependant, la plupart des équipes s'en servent car cela facilite le partage du travail.

Une tâche est un petit travail qui contribue à finir une story. Toutes les activités de développement sont déroulées lors d'un sprint : on identifiera donc les tâches à partir de ces activités.

Pour chaque story avec du code, on retrouve des tâches similaires : concevoir, coder l'IHM, coder la couche métier, faire les tests unitaires...

La définition de fini, avec sa liste de vérifications, aide à identifier les tâches.

Si l'essaimage est pratiqué, l'identification des tâches sera effectuée localement par chaque essaim, suivie d'une restitution collective.

Existe-t-il des tâches sans story ?

Il y a quelques années, j'avais l'habitude d'utiliser deux catégories de tâches qui n'étaient pas associées à des stories : les tâches « urgentes » pour un travail ajouté pendant le sprint, et les tâches « récurrentes » pour un travail qui revenait à chaque sprint.

Exemple : le travail à faire pour déployer le logiciel à chaque fin de sprint sur un serveur de test est récurrent, celui pour répondre à un client chez qui le produit ne fonctionne plus est urgent.

J'ai évolué depuis : plutôt que créer des tâches non associées à une story, je préfère créer une story correspondant à ces travaux, cela se voit mieux. Le travail récurrent va dans une story de finition de sprint, le travail urgent est placé dans une story de réserve, où seront placées les urgences subies, en tant que tâches.

La liste des tâches constituée lors de la réunion de planification n'est pas figée : des tâches pourront être ajoutées, d'autres supprimées et d'autres décomposées pendant le sprint. Sur des sprints longs, il est possible, si nécessaire, de faire une seconde réunion de planification plus tard pour compléter et revoir le plan. Les stories restent ordonnées dans le sprint : celles moins prioritaires seront réalisées en fin de sprint, ce qui laisse du temps pour affiner leur décomposition en tâches.

L'équipe Peetic, cinq personnes et des sprints de deux semaines, identifie en moyenne une quarantaine de tâches.

9.2.5. S'engager

L'engagement est une valeur de Scrum, mais sur quoi porte-t-il ?

Disponibilités pour ce sprint

D'abord, on fait le point sur les prévisions de participation. Dans le cas où des développeurs ne sont pas à plein temps pour le sprint qui commence, il est utile de le savoir, car cela peut impacter l'engagement.

On va noter les jours où des experts viennent aider l'équipe. Pour s'y retrouver, on peut élaborer un calendrier visuel^[11].

Le ScrumMaster sera vigilant sur les dépendances possibles entre des membres de l'équipe, ou sur des experts. Lorsque des compétences ne sont pas partagées, cela peut provoquer des goulets d'étranglement. Pour les anticiper, on peut revenir sur l'ordre des stories.

Pas d'engagement sur une liste de stories

Il est préférable que l'engagement de l'équipe ne porte pas sur un nombre exact de stories, ni sur une capacité en points. En effet, cela mettrait l'équipe dans une situation de périmètre fixé pour une date fixée (avec un effectif stable) et on sait que ne pas avoir de variable d'ajustement dans les plans est une des raisons de l'accumulation de dette technique.

Il faut garder des stories comme variable d'ajustement du sprint.

Un engagement collectif est tout de même important pour créer les conditions de motivation de l'équipe. Il permet de déceler les réticences chez des développeurs, qu'il est préférable de prendre en compte avant de finir la réunion. Il portera sur l'objectif du sprint.

Engagement sur l'objectif du sprint

L'objectif est énoncé en une phrase qui sert à canaliser les énergies. Il est élaboré par l'équipe, à partir d'une proposition du Product Owner.

L'objectif du sprint décrit le bénéfice attendu à la fin du sprint, sur lequel s'engage l'équipe et qui détermine le succès du sprint.

Il porte le plus souvent sur un domaine fonctionnel, une feature. Cependant, au début du projet, lors des premiers sprints de la première release d'un produit, l'objectif peut être orienté sur des considérations plus techniques que fonctionnelles pour lever des risques importants.

Exemples d'objectif de sprint :

- sprint 1 – authentification des utilisateurs.
- sprint 2 – ouvrir le magasin de vente en ligne avec la nourriture pour animaux.
- sprint 3 – permettre le paiement par Paypal.

Attention à ne pas définir cet objectif comme un énoncé de plusieurs stories, il ne faut pas abuser des « et », et si possible les éviter.

L'objectif du sprint est annoncé à toutes les parties prenantes. Il est d'ailleurs souhaitable de le préparer avec eux à la fin de la revue du sprint précédent. Cependant c'est l'équipe qui le finalise à ce moment de la réunion, en connaissant bien le travail à faire.

Quel sens donner à l'engagement de l'équipe ?

L'engagement correspond, dans une situation donnée, aux conditions dans lesquelles la réalisation d'un acte ne peut être imputable qu'à celui qui l'a réalisé [Joule, Manipulation].

Sur l'objectif du sprint, l'équipe s'engage à produire un incrément avec le niveau de qualité requis dans la définition de fini. Elle s'engage également à être transparente sur l'avancement du sprint et à prévenir le Product Owner dès que l'objectif est remis en question, ou s'il peut être augmenté.

Pour que l'engagement ait un sens, il est indispensable qu'il soit librement consenti, que l'équipe ne subisse pas de pression et qu'il soit revu si les conditions changent.

9.2.6. Lancement du sprint

Prendre des tâches

Ce sont les membres de l'équipe qui prennent eux-mêmes les tâches, en respectant l'ordre des stories. Il n'est pas utile d'aboutir à l'attribution de toutes les tâches : il suffit que

chacun ait du travail pour les premiers jours du sprint ; l'affectation des autres tâches est différée, elles seront prises pendant le sprint en fonction des disponibilités de chacun.

En général, la prise d'une tâche ne pose pas de problème dans l'équipe. L'essaimage permet d'éviter les tergiversations.

Et si un travail pénible mais important n'est pris par personne ?

Dans les développements traditionnels, certaines tâches sont considérées comme ingrates, par exemple faire un plan de test, du *reporting* et autres documentations, et il est parfois difficile de trouver un volontaire. C'est en général le chef de projet qui les affecte de façon autoritaire.

J'ai côtoyé de nombreuses équipes Scrum et je n'ai jamais constaté le phénomène des tâches pénibles que personne ne voudrait prendre.

Il y a plusieurs raisons pour que ça ne se produise pas :

– **Il n'y a pas de tâches pénibles** – En effet, le découpage en tâches est fait de façon radicalement différente. Lors de la réunion de planification, les tâches sont identifiées à partir des stories sélectionnées et servent clairement à finir quelque chose. Il n'y a pas de tâches non corrélées aux résultats concrets, comme par exemple lorsqu'on demande à quelqu'un d'écrire à la fin du projet des jeux de tests unitaires simplement parce que c'est demandé dans le processus.

– **Si une tâche est malgré tout considérée comme pénible, elle sera courte** – En effet, une tâche dans un sprint représente moins d'une journée de travail pour une personne. Cela sera perçu comme beaucoup moins pénible que, par exemple, la tâche d'écriture d'une documentation de conception pendant dix jours dans une approche traditionnelle. L'affectation peut d'ailleurs se faire de manière ludique et collaborative s'il n'y a pas de volontaire : pile ou face, avec des dés, ou en binôme.

– **Le passage d'un mode directif à un mode autonome responsabilise chacun** – Le fait que les tâches soient identifiées en commun pousse à les trouver moins ingrates que si elles sont imposées.

Début du travail

On connaît l'ordre des stories. Ce sont les tâches associées à ces stories qui sont commencées en premier. On se retrouvera demain à la mêlée.

9.3. Résultats de la planification du sprint

Le résultat essentiel est l'équipe focalisée vers l'objectif. La phrase qui le décrit est affichée de façon à rester visible à toute l'équipe pendant la durée du sprint.

Il est également communiqué aux parties prenantes qui sauront de cette façon ce que fait l'équipe et à quoi elle s'est engagée.

Chaque story est partie dans le sprint après vérification de sa définition de prêt. Sa définition de fini pourra être enrichie pendant le sprint et servira pour vérifier sa finition. Elle est donc un outil de communication entre le PO et l'essaim qui développe la story.

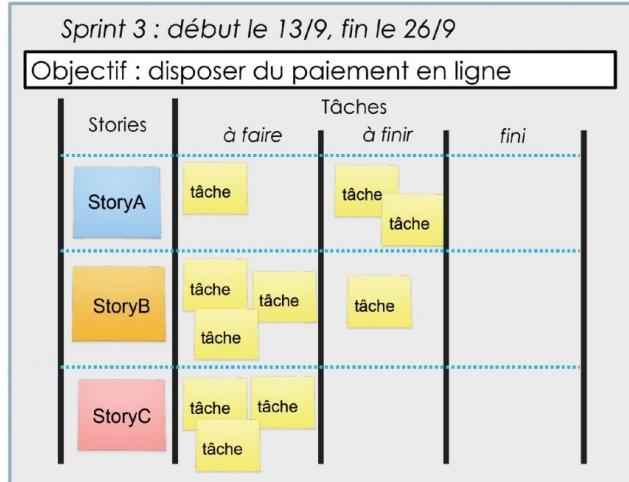


Fig. 9.4 Le tableau

Le plan de sprint contient la liste des stories avec leurs tâches, sous une forme visible et facilement accessible. Une tâche est décrite simplement :

- **Un nom et la description du travail à faire** – Il suffit d'avoir un nom permettant d'identifier la tâche et, éventuellement, du texte collecté lors de la réunion permettant de comprendre le travail à faire.
- **La personne qui prend la tâche** pour la réaliser – Une tâche peut être réalisée par une ou plusieurs personnes.

9.4. La « *planif* » sur le terrain

9.4.1. Attention à ne pas mettre trop de pression sur l'équipe

Ce n'est pas le ScrumMaster qui décide de ce qui doit être fait. Ce n'est pas non plus le Product Owner qui impose à l'équipe le périmètre du sprint en lui donnant la liste des stories à réaliser.

Le Product Owner **propose** l'objectif et définit l'ordre des stories candidates pour le sprint. C'est l'équipe qui **décide**.

Un Product Owner peut avoir tendance à insister pour avoir plus. Ce n'est pas une bonne pratique : cela pousse à un engagement subi, qui est contre-productif.

En revanche, les discussions peuvent amener l'équipe à proposer des changements dans les priorités pour faire passer une story non prévue à la place d'une autre dans le sprint. C'est au PO d'accepter ou pas.

De mon point de vue, l'engagement sur l'objectif est préférable à l'engagement « subi » sur un nombre de points, encore pire quand l'estimation est faite pendant la planification de sprint.

9.4.2. Pas de tâches préparées à l'avance

Des ScrumMasters soucieux d'efficacité après une réunion de planification de sprint trop longue peuvent décider, la fois suivante, d'arriver avec une liste des tâches déjà prête. Il est possible que la réunion soit effectivement raccourcie, mais cela a l'inconvénient énorme de moins impliquer l'équipe. En effet, si les tâches sont déjà identifiées, et pire affectées, l'équipe va se sentir moins responsabilisée. Ce serait un retour à un schéma de gestion de projet avec un chef :

- le chef de projet prépare le plan tout seul (au mieux il le montre aux membres de l'équipe) ;
- il le montre à son management qui le critique ;
- le chef de projet ajuste le plan qui s'applique.

La pratique Scrum d'une équipe autonome et responsabilisée rend caducs ces allers-retours : tout est fait collectivement, en une seule unité de temps et de lieu.

9.4.3. Faire de la conception

Une erreur classique des équipes novices en Scrum est de se lancer à corps perdu dans la réalisation des stories. Le développement de logiciel nécessite de la réflexion. Bien que Scrum n'évoque pas explicitement de pratiques d'ingénierie, il est évidemment nécessaire de faire de la conception. Avec les méthodes agiles, la conception n'est pas faite une fois pour toutes au début du projet, elle émerge tout le temps. Chaque sprint comporte des activités de conception.

La réunion de planification de sprint n'est pas le moment idéal pour faire de la conception détaillée. Cependant, l'identification des tâches nécessite de réfléchir à la façon dont une story va être conçue. Les discussions qui s'engagent permettent de partager la connaissance de cette conception entre les membres de l'essaim.

Les activités de conception peuvent commencer pendant l'affinage d'une story et se poursuivre pendant le sprint. Chaque essaim s'occupe de la conception de sa story.

9.4.4. Décomposer en stories et tâches courtes

Dans la plupart des organisations, avant de passer à Scrum, la gestion des projets se base sur des tâches plus longues que celle conseillée dans les sprints : des tâches de plusieurs jours, voire plusieurs semaines.

Avec Scrum, une tâche prend en moyenne moins d'un jour et une story un à trois jours. Un bon principe est d'avoir une tâche finie le lendemain du jour où elle a été commencée. C'est un constat à vérifier lors de la mêlée quotidienne.

Cela veut dire que non seulement l'effort pour réaliser la tâche est limité, mais que cet effort est condensé sur un jour au lieu d'être morcelé sur plusieurs jours. Cela donne la satisfaction de finir vite les choses, c'est important pour le moral.

9.4.5. Garder du mou dans le plan

Des événements inattendus viennent toujours freiner l'avancement du sprint, bloquant ou ralentissant une ou plusieurs tâches en cours. Pour empêcher ces événements de remettre en cause l'engagement de l'équipe, il faut garder du mou lorsqu'on planifie le sprint.

La façon la plus simple est de prendre un engagement sur l'objectif.



Fig. 9.5 Il ne faut pas hésiter à mettre du mou dans les plans

9.4.6. Prendre un engagement raisonnable

Même sans la pression du Product Owner, une équipe qui débute avec Scrum a tendance, par excès d'optimisme, à prendre plus de stories que ce qu'elle peut raisonnablement réaliser en un sprint.

S'il peut arriver que l'engagement ne soit pas tenu, cela ne doit plus arriver après plusieurs sprints. Une équipe va apprendre à progresser. Cet apprentissage passe par la mise en place de pratiques correctives lors de la rétrospective et par un engagement raisonnable.

Bien commencer

La valeur Scrum

L'engagement dans un sprint ne doit pas imposer une obligation contre nature qui produirait inexorablement de la dette technique, en plus de la frustration. C'est un engagement au sens tentative à faire de son mieux.

La question à se poser

Est-ce que l'objectif du sprint est raisonnable ?

L'équipe mélange stories et tâches.

L'équipe estime les tâches en heures.

De mauvais signes

La fin de réunion de planification est bâclée, sans prise d'engagement.

Par quoi démarrer

Préparer un beau tableau avec des Post-it®.

Une lecture pour tous

Petit traité de manipulation à l'usage des honnêtes gens, [Joule, *Manipulation*], un livre qui aide à éviter de se faire manipuler.

À retenir

La planification du sprint est la première réunion du sprint, celle qui permet à l'équipe de s'engager sur l'objectif du sprint et s'accorder sur les stories associées.

C'est aussi la plus délicate : son bon déroulement conditionne le succès du sprint. Pour la réussir, il convient de ne pas la voir uniquement comme une séance de planification, mais aussi comme un exercice collectif pendant lequel l'équipe apprend à s'auto-organiser et à partager la connaissance sur le produit et sur la solution pour le développer.

Le management visuel et l'essaimage sont deux techniques efficaces qui contribuent au succès de cette réunion, et donc du sprint.

Références :

☞ R.-V. Joule et J.-L. Beauvois, *Petit traité de manipulation à l'usage des honnêtes gens*, 2014, PUG.

☞ Jeff Sutherland et Ken Schwaber, *Scrum Guide*, 2013, VF.

<http://wiki.ayeba.fr/Guide+Scrum+2013>

Notes

[1] Comme le calendrier visuel de Bruno : <http://www.aubryconseil.com/post/Le-calendrier-visuel>.

La mêlée quotidienne

Il y a des grands projets pour lesquels on apprend un jour, en écoutant la radio le matin, qu'ils ont subitement des mois de retard.

Frederic Brooks, l'auteur de *Mythical Man Month*, à qui on demandait comment fait un projet pour avoir un an de retard, répondait il y a plus de vingt ans « *En prenant un jour de retard, puis un autre...* »^[1].

C'est pour en prendre conscience et réagir que la mêlée a lieu tous les jours.

La mêlée au rugby représente la quintessence de l'effort collectif. La mêlée avec Scrum est plus un moment de partage que d'effort.



Fig. 10.1 La mêlée, un grand moment de fraternité.

Ce chapitre présente cette pratique très représentative de Scrum. C'est d'ailleurs la plus utilisée, d'après les enquêtes et les retours du terrain.

Parfois on découvre une équipe qui dit appliquer Scrum, mais qui, en fait, pratique uniquement une réunion quotidienne pour faire l'avancement.

Nous verrons que le suivi de l'avancement n'est pas l'objectif principal de la mêlée, qui est de développer la collaboration dans l'équipe.

10.1. Suivre l'exécution du sprint

Programmée tous les jours, la mêlée rythme l'exécution du sprint qui consiste à faire ce qui a été défini lors de la planification.

Mais il se produit toujours des aléas : « *aucun plan ne survit au contact de l'ennemi* ». On peut identifier deux formes d'imprévus : ceux qui sont inhérents à l'exécution du sprint par l'équipe, appelés des obstacles, et ceux qui sont dus à des changements exogènes dans le plan de départ.

10.1.1. Obstacle

Un obstacle bloque ou ralentit le bon déroulement du sprint.

Sébastien, développeur de l'équipe Peetic, n'arrive pas à commiter ses modifications sur Git ; Laurent, l'expert cartographie, s'est cassé le bras au ski pendant le week-end.

Un obstacle est identifié au cours de l'exécution du sprint, dès qu'il apparaît. Il est décrit par son état (identifié, en cours de résolution, résolu), son impact (défini par les tâches qui sont bloquées ou freinées) et la date de sa découverte.

Les obstacles sont visibles dans le tableau des obstacles, à côté du tableau du sprint (figure 10.2).

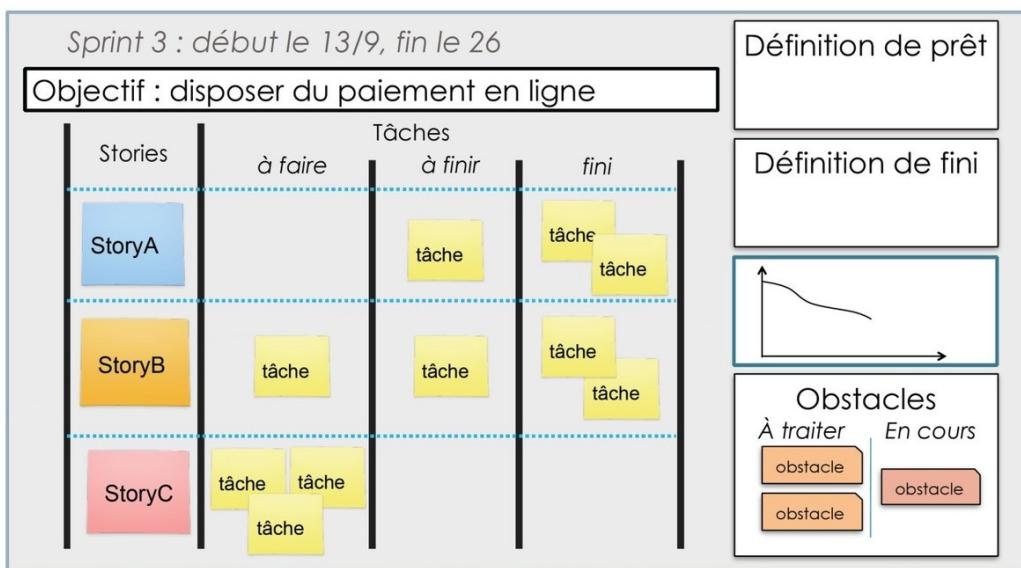


Fig. 10.2 Le tableau avec les obstacles

Le ScrumMaster a pour mission de les éliminer. Cela peut nécessiter du travail pour l'équipe. Une fois résolu, il faudra réfléchir à des moyens pour éviter qu'il se reproduise, si c'est dans le pouvoir de l'équipe.

10.1.2. Perturbations exogènes

En principe, l'équipe n'est pas perturbée pendant le sprint. Le ScrumMaster la protège des interférences extérieures.

Néanmoins, cela peut arriver, avec deux types d'événements :

- l'ajout de travail à l'équipe,
- la sortie d'un membre de l'équipe pendant quelque temps.

Dans les deux cas, l'objectif du sprint peut être revu et modifié par l'équipe, au sortir de la perturbation.

La transparence pousse à mettre le travail ajouté sur le tableau de sprint. Le prélèvement ponctuel d'une personne aura sa place dans la liste des obstacles.

Si l'équipe constate que du travail « urgent » est régulièrement ajouté à chaque sprint, elle pourra anticiper la perturbation en ajoutant une story de réserve. Si l'équipe constate qu'on lui retire de la capacité à chaque sprint, elle en tiendra compte lors de son engagement.

10.1.3. Terminaison des travaux

En dehors de ces aléas, le tableau de sprint est ajusté quand des tâches sont finies, et le bac d'arrivée est mis à jour quand des stories se terminent. Une story est déclarée finie quand sa définition de fini est vérifiée. C'est en principe le PO qui est le responsable de cette action. Mais si l'accord sur les conditions s'est fait avec l'équipe, et que la confiance est là, il peut déléguer cette responsabilité.

10.1.4. Changements dans les tâches

L'équipe a tout loisir d'ajouter des tâches dans le plan, ou d'en supprimer.

Il arrive que des tâches soient découvertes pendant le sprint : en travaillant sur une story, on s'aperçoit qu'un travail à faire important a été oublié lors de la planification du sprint.

Une fois la story finie, les tâches peuvent être supprimées, il ne sert à rien d'en garder une trace.

10.2. Une réunion quotidienne

La réunion quotidienne se nomme le « *Daily Scrum Meeting* » en anglais. La Scrum Alliance utilise simplement *daily scrum*.

En France, on entend fréquemment « daily » et parfois DSM. C'est vrai qu'au nord de Montauban, appeler une réunion mêlée a un côté peu rassurant pour les participants et ne contribue pas à promouvoir l'agilité auprès des développeuses.

Cette réunion s'appelle **stand up meeting** dans *Extreme Programming*, ce qui fait bien comprendre qu'on reste debout.

10.2.1. Pour réussir le sprint

La mêlée est plutôt facile à décrire : un point de rencontre où tous les membres de l'équipe répondent à trois questions simples. Cela pourrait laisser à penser, à tort, qu'il s'agit de vérifier ce que font les gens.

En fait, elle sert à optimiser la probabilité que l'équipe atteigne l'objectif du sprint. La mêlée est de « l'inspection et de l'adaptation » au quotidien.

L'adaptation consiste, le plus souvent, à fixer les réunions ou discussions d'équipe nécessaires dans la journée pour éliminer les obstacles ou finir des stories.

10.2.2. La mêlée appartient à l'équipe

Toute l'équipe participe à la réunion, y compris le Product Owner. Le ScrumMaster s'assure que la réunion a lieu et que les règles sont respectées. Pendant la réunion, il n'a pas de responsabilité particulière : la mêlée est un des moyens permettant d'aller vers plus d'auto-organisation, ce n'est pas une réunion pour faire un compte-rendu au ScrumMaster.

Les personnes extérieures à l'équipe peuvent y assister, mais elles ne participent pas à la discussion.

La distinction entre l'équipe et les parties prenantes rappelle que seuls les membres de l'équipe sont réellement engagés dans l'objectif du sprint.

Ceux qui sont engagés savent la meilleure attitude avoir devant une situation donnée, pas ceux qui sont simplement concernés. Cela est illustré dans l'histoire de la poule et du cochon.

Les amateurs de rugby (ou de sport collectif) comprendront mieux cette devise : « *Pendant le match, le jeu appartient aux joueurs* ».

10.2.3. Un cérémonial balisé

La mêlée repose sur des pratiques éprouvées dans les dimensions temps et espace.

Lieu : l'endroit idéal pour la tenir est devant le tableau mural où est affiché le travail de l'équipe.

Le plan de sprint est actualisé au fil de l'eau : quand un membre de l'équipe finit une tâche, il le met à jour (et le signalera à la mêlée suivante).

Si toutes les personnes de l'équipe ne sont pas dans le même espace physique, on utilise des outils de vidéoconférence.

Durée : la mêlée dure moins d'un quart d'heure.

Fréquence : la mêlée est quotidienne.

■ L'équipe Peetic fait sa mêlée tous les jours à 9 h 15 devant son tableau.

Le succès de l'application de Scrum sur un projet passe par le respect des principes de base. Un de ceux-ci peut se résumer en « *no scrum no win* », comme pour le rugby : une équipe qui ne fait pas tous les jours la mêlée n'est probablement pas vraiment constituée en tant qu'équipe.

La mêlée Scrum repose sur un rituel simple et précis : tous les jours, debout, un quart d'heure et pas d'interruptions de parties prenantes.

Il ne s'agit pas d'être dogmatique, il ne me paraît pas dramatique de sauter un jour, de dépasser 15 minutes ou de faire participer un expert, si cela reste exceptionnel et que ça va bien dans l'équipe. Le premier et le dernier jour du sprint sont particuliers : il y a d'autres réunions du sprint et la mêlée n'est pas faite ces jours-là.

Le rituel de la mêlée est orienté vers les gens. Cependant, nous verrons ensuite une nouvelle façon de les mener, orientée *stories*.

10.3. La mêlée classique

L'équipe se **réunit**. Chacun s'exprime à tour de rôle sur les **trois questions** : ce qu'il a fait, ce qu'il va faire et ce qui l'empêche d'avancer.

En fonction de ces informations, l'équipe **s'adapte** en s'organisant pour éliminer les obstacles et atteindre l'objectif du sprint.

10.3.1. Se réunir

La réunion s'effectue avec toutes les personnes de l'équipe, debout en demi-cercle face au tableau mural. Il est préférable que la mêlée ait lieu : le matin en arrivant, tous les jours au même endroit et à la même heure. La réunion commence à l'heure prévue, même si des personnes sont en retard.

Si la réunion a lieu dans un endroit où les postes de travail sont accessibles, le ScrumMaster doit veiller à ce que les occupants du bureau enlèvent les mains de leur clavier et ne gardent pas un œil sur l'écran pendant la réunion. De même, l'usage des messageries instantanées, des smartphones, de Facebook ou Twitter est déconseillé pendant un quart d'heure...

Pour tenir les objectifs de la mêlée dans un délai aussi court, il est souhaitable de disposer d'une aide visuelle. Le tableau de sprint fournit cette assistance.

Si on utilise un outil, on peut s'en servir, lors de ces réunions, pour mettre à jour l'état des tâches. Avec un vidéo-projecteur, on projette la liste des tâches sur un écran et il est possible, éventuellement, de les changer de colonne en direct. Cependant, il vaut mieux réservé l'outil aux équipes dispersées : cette pratique complique la logistique et a comme inconvénient qu'une seule personne a les mains sur le clavier. Cela ne contribue pas à renforcer la responsabilité de l'équipe.

10.3.2. Répondre aux trois questions

Prenons l'hypothèse que la mêlée ait lieu le matin, ce qui est majoritairement le cas.

Présenter ce qui a été fait

Chaque participant répond à la première question :

« *Qu'ai-je fait hier ?* »

Il s'agit, pour chaque membre de l'équipe, de parler des tâches sur lesquelles il a travaillé. Il indique si la tâche est encore en cours ou si elle est finie.

Rappelons que le Product Owner et le ScrumMaster sont aussi des membres de l'équipe.

Prévoir ce qui va être fait

Chaque participant répond à la deuxième question :

« Que vais-je faire aujourd'hui ? »

Il présente les tâches sur lesquelles il prévoit de travailler (en principe, il n'y en a qu'une ou deux). Pour chacune d'elles, il indique en quoi elle consiste et s'il pense la finir dans la journée.

Le découpage doit être suffisamment fin pour qu'une tâche puisse être finie en une journée de travail.

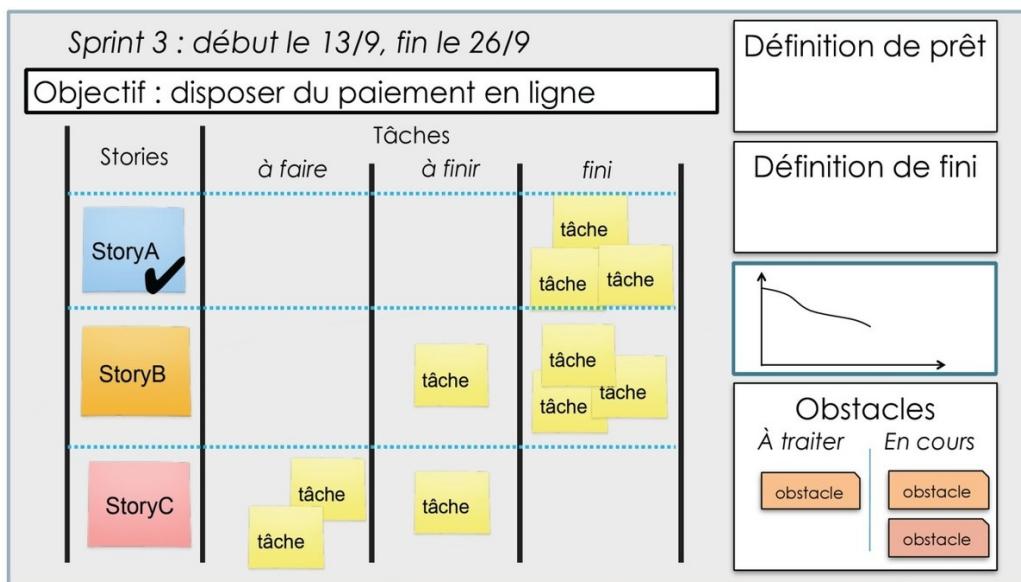


Fig. 10.3 Évolution du tableau (une story est finie).

En quelque sorte, lors de la mêlée, chacun s'engage devant ses pairs. Lorsqu'une personne dit ce qu'elle va faire, elle l'annonce devant toute l'équipe, c'est sa parole.

Identifier les obstacles

Chaque participant répond à la troisième question :

« Quels sont les obstacles qui me freinent dans mon travail ? »

On n'attend pas la mêlée pour signaler un obstacle franc, la question s'adresse donc aux obstacles plus subtils.

C'est au ScrumMaster de rechercher les vrais obstacles derrière les formulations vagues. Nicolas, le SM de l'équipe Peetic, face à David annonçant qu'un problème avec Céline, le PO, l'empêche de finir la story 22, a réussi à lui faire reformuler en « j'ai proposé deux façons de procéder pour l'arrangement des

boutons sur la fenêtre de validation de la story 22 et j'attends toujours la réponse de Céline ». Ce qui rend l'identification de l'obstacle et son élimination plus faciles...

Quand un obstacle est identifié en séance, il est possible qu'un des membres de l'équipe connaisse déjà un moyen de l'éliminer. Il le donne immédiatement.

En revanche, si des personnes ont seulement des pistes de solution, elles auront tendance à lancer une discussion, mais ce n'est pas le lieu adéquat pour la prolonger. Le ScrumMaster propose aux personnes intéressées de continuer la discussion dans la journée.

10.3.3. S'adapter pour l'atteinte des objectifs

Comme la mêlée au rugby vient après une phase de désordre et permet au jeu de repartir sur de nouvelles bases, la mêlée quotidienne permet à l'équipe de se synchroniser par rapport à l'objectif du sprint.

C'est un exercice quotidien d'inspection et d'adaptation.

Avec les informations collectées sur les stories et les obstacles, l'équipe acquiert une connaissance objective de l'avancement. L'adaptation, si elle est jugée nécessaire, peut se manifester de plusieurs façons : re-négocier l'objectif du sprint, ajouter ou enlever une story au sprint, revoir les règles de fonctionnement, comme celles de l'essaimage.

En fin de sprint, dans le cas où l'équipe a un peu de temps mais pas suffisamment pour prendre une nouvelle story, elle se consacre à l'amélioration de la qualité. Il y a toujours à faire, par exemple du remaniement de code ou la mise à jour d'un document pour réduire la dette technique.

10.4. La mêlée orientée stories

Cette technique correspond un peu à ce qu'est le maul^[2] par rapport à la mêlée au rugby : la participation active se fait selon la situation du moment.

Cette façon de procéder est adaptée aux équipes qui pratiquent l'essaimage, présenté dans le chapitre 9 *Planifier le sprint*. La différence porte sur la façon de répondre aux trois questions.

On **examine** les stories dans l'ordre. Pour chaque story, un membre de son essaim présente ce qui a été fait, ce qui va être fait et les obstacles qui empêchent de finir la story. On ajuste la **répartition** sur les stories.

10.4.1. Examiner les stories

Dans cette approche, les trois questions ne sont pas pour chaque membre de l'équipe, mais pour chaque story en course.

Le référent de la story, s'il y en a un, est bien placé pour présenter les informations permettant de connaître son avancement. Il donne les conditions d'acceptation qui sont vérifiées depuis la mêlée précédente, et celles qui devraient l'être d'ici la prochaine. Avec

les autres développeurs qui participent à la réalisation de la story, un obstacle éventuel est identifié et, si nécessaire, une discussion est planifiée pour l'éliminer.

Puis on passe à la story suivante. On traite de cette façon uniquement les stories sur lesquelles il y a eu de l'activité depuis la mêlée précédente.

10.4.2. Se répartir sur les stories

La répartition de l'équipe en essaims est ajustée. Quand un développeur a fini une tâche et qu'il n'est pas référent de la story, des questions se posent : reste-t-il avec cet essaim, va-t-il aider un autre essaim ou bien ouvre-t-il une nouvelle story ? Les réponses sont fournies après discussion.

Par ailleurs, il convient de définir à quel moment de la journée les interventions du Product Owner (pour vérifier des conditions d'acceptation, par exemple) ou du ScrumMaster (pour aider à éliminer un obstacle) seront les plus pertinentes pour chaque essaim qui en a besoin.

La suite de la mêlée est identique à l'approche par personne.

10.4.3. Risques et bénéfices

Comme l'objectif du travail d'une équipe est de finir des stories, le bénéfice de cette approche est une meilleure focalisation sur leur achèvement.



Fig. 10.4 L'essaimage peut améliorer la vitesse

A contrario, il existe le risque qu'une personne ne s'exprime pas, si les référents de story ne laissent pas la parole à tous les participants à l'essaim.

La mêlée permet d'identifier des dépendances entre les tâches et de suivre leur état. Si l'objectif primaire est de finir les tâches, l'objectif essentiel est de finir les stories. Pour qu'une story soit finie, il faut que toutes les tâches associées soient terminées. Cela conduit à avoir des priorités pour la prise des tâches libres. Les discussions lors de la mêlée permettent à l'équipe de se mettre d'accord sur l'ordre dans lequel les tâches doivent être faites.

10.5. Les informations utiles au quotidien

10.5.1. L'humeur de l'équipe

Le résultat essentiel est une équipe qui s'est organisée pour que son travail du jour contribue au succès du sprint. La réunion quotidienne permet à l'équipe de lancer une journée en ayant préparé son organisation. Le ScrumMaster peut se faire une idée de la santé de l'équipe par sa perception des humeurs de chacun ou en demandant que le ressenti quotidien soit exprimé.

10.5.2. Les indicateurs de sprint

Le *burndown chart* de sprint est un graphique, actualisé tous les jours, montrant la tendance du reste à faire. On peut choisir une de ces variantes :

- le burndown en tâches,
- le burndown en stories.

Ceux qui préfèrent les courbes qui montent à celles qui descendent choisiront la représentation avec un *burnup chart*.

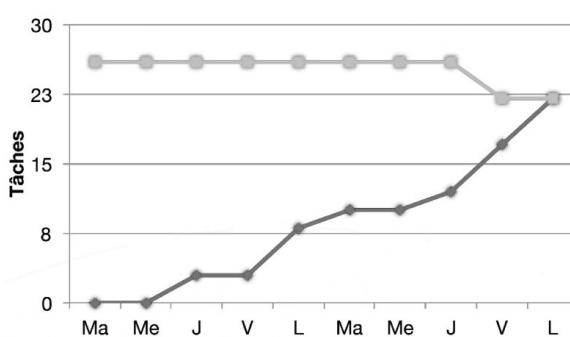


Fig. 10.5 Burnup en tâches

Ces indicateurs sont destinés uniquement à l'équipe. Après quelque temps, elle peut décider de les abandonner, car ils ne sont pas d'une grande utilité pour qui a l'habitude d'analyser les tableaux de sprint.

10.5.3. La liste des obstacles

C'est la responsabilité du ScrumMaster de classer les obstacles par priorité et de faire en sorte qu'ils soient éliminés au plus vite. Certains peuvent être du ressort de l'équipe, d'autres ne peuvent avoir une solution qu'avec l'intervention de personnes extérieures, dans d'autres équipes (par exemple, si une équipe support s'occupe de l'environnement de développement) ou au niveau de la direction du projet.

Les obstacles résolus sont conservés au moins jusqu'à la fin du sprint : la liste des obstacles est utile lors de la rétrospective.

10.6. La mêlée sur le terrain

10.6.1. S'en tenir à un quart d'heure

Pour que la mêlée ne dure pas trop longtemps, il faut suivre les règles : elle a lieu tous les jours de travail, les solutions pour éliminer les obstacles ne sont pas discutées en réunion, seuls les membres de l'équipe peuvent parler.

Un accessoire efficace est la « boîte à meuh », actionnée dès qu'une discussion sort du cadre de la réunion. On peut aussi utiliser le coucou pour signaler la fin du temps de parole.

10.6.2. Ce n'est pas un compte-rendu au ScrumMaster

La mêlée ne doit pas être l'occasion pour le ScrumMaster de reprendre les habits du chef de projet. Les développeurs s'adressent à toute l'équipe, pas seulement à lui.

Il n'assigne pas les tâches : c'est à l'équipe et en particulier à l'essaim de mettre en œuvre l'auto-organisation, son seul rôle est d'y contribuer.

Quand les réunions sont faites avec un outil plutôt que devant un tableau physique, le risque est encore plus grand : bien souvent, c'est le ScrumMaster seul qui va manipuler l'outil.

Le ScrumMaster n'est pas non plus une petite main qui fait les travaux que les membres de l'équipe n'ont pas envie de faire.

10.6.3. Attention à ne pas mesurer le temps passé

Même s'il est utile de bien estimer, c'est encore plus important de finir les choses. C'est pourquoi Scrum ne s'intéresse pas au temps passé ni sur les tâches, ni sur les stories. Pour certains, c'est un changement radical.

Chez un de mes clients, un développeur à qui on demandait combien il restait à faire sur la tâche qu'il venait d'évoquer, consulta la liste et y lut le nombre d'heures inscrit la veille pour cette tâche, soit 14 heures. Il réfléchit brièvement et dit : j'ai travaillé 2 heures sur cette tâche alors il reste à faire 14 moins 2 soit 12 heures.

La mesure du temps passé sur une tâche n'est pas utile avec Scrum, pas plus que son estimation. Cela évite de faire du micromanagement avec un suivi individuel.

La mesure du temps passé sur une story n'est pas faite non plus.

10.6.4. Varier les mêlées

Il est fondamental que la mêlée reste un moment où l'équipe reprend de l'énergie pour la journée. Pour éviter de tomber dans la routine, le ScrumMaster (ou un membre de l'équipe) propose des variations dans le déroulement des mêlées :

- la prise de parole se fait un coup de gauche à droite, un coup de droite à gauche, ou de façon aléatoire ;
- la réponse aux trois questions se fait une fois à la suite pour chaque personne, une autre fois par trois tours avec une question à chaque fois.
- des accessoires permettent de renforcer la cohésion de l'équipe, par exemple :

- de la musique annonce la mêlée qui change à chaque sprint ;
- un ballon de rugby passé de main en main pour montrer qui a la parole.

Bien commencer

La question à se poser

Est-ce que les gens vont à la mêlée quotidienne avec plaisir ?

De mauvais signes

Des gens sont assis.

Les obstacles ne sont pas collectés.

On n'en sait pas plus sur la tenue de l'objectif du sprint et sur la finition des stories en sortant de la mêlée.

Par quoi démarrer

Se mettre debout devant le tableau.

Une lecture pour tous

Le dictionnaire amoureux du rugby [Herrero, Rugby] pour son lyrisme sur la mêlée.

À retenir

La mêlée est une réunion qui se passe tous les jours, avec toute l'équipe debout pour faire le point sur le travail effectué et celui qui reste à faire d'ici la fin du sprint. C'est une pratique qui a prouvé son efficacité pour améliorer l'auto-organisation et qui ne coûte pas cher à mettre en place.

Il ne s'agit pas simplement de rester debout et de répondre aux trois questions. L'objectif est de mettre l'équipe en situation pour que l'exécution du sprint soit un succès. Pour cela, les obstacles sont identifiés et les rendez-vous pris pour les éliminer dans la journée.

Références :

☞ Daniel Herrero, *Dictionnaire amoureux du rugby* – version illustrée, 2015, Flammarion.

Notes

[1] <http://courses.cs.vt.edu/~cs1104/HLL/Brooks.html>, voir [page 153].

[2] Un maul commence lorsqu'un joueur portant le ballon est saisi par un ou plusieurs adversaires et qu'un ou plusieurs coéquipiers du porteur du ballon se lient à ce dernier.

La revue de sprint

Avant d'assister aux revues de sprint, j'ai participé à de nombreuses revues de projet. Elles portaient sur des documents, qui devaient être envoyés plusieurs jours avant la réunion afin que les participants aient le temps de les lire. La réunion elle-même pouvait durer longtemps. Une revue de fin de phase d'un développement de la partie sol d'un système spatial peut durer plusieurs jours, chaque document étant épluché et chaque remarque commentée.

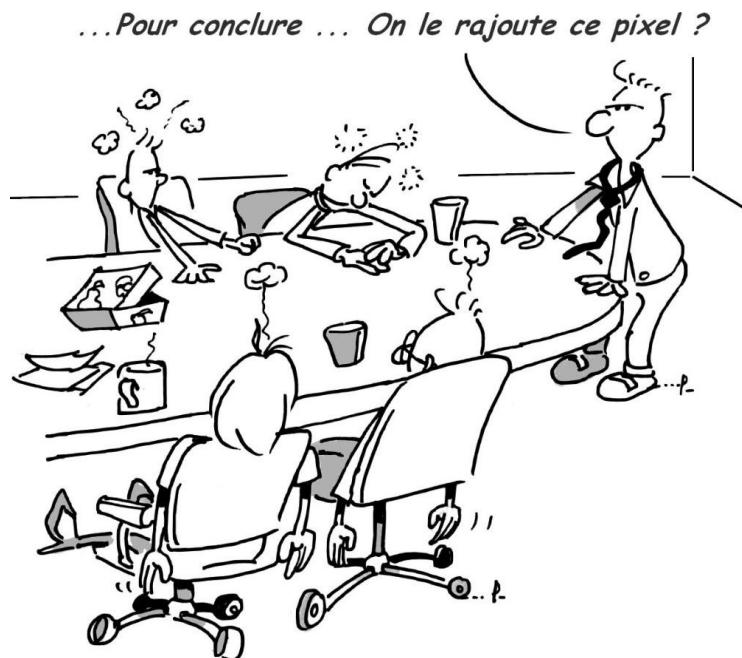


Fig. 11.1 Une revue à l'ancienne.

L'objectif de ces revues était de connaître l'état d'un projet pour prendre des décisions sur sa poursuite. La revue de sprint Scrum a le même but, mais la façon d'y arriver est radicalement différente.

Scrum est en adéquation avec le *Manifeste agile*, qui dit que « *pour connaître l'avancement d'un développement, il vaut mieux se baser sur du logiciel opérationnel plutôt que sur de la documentation* ». Dans un développement de logiciel, on va montrer du code qui marche.

Ce principe est mis en œuvre lors de la revue, par la démonstration de l'incrément de produit réalisé pendant le sprint. C'est un moment essentiel de la mise en œuvre de la théorie empirique de Scrum : la démonstration permet d'avoir une **visibilité** sur le produit, et le **feedback** favorise son adaptation aux besoins.

11.1. Plus qu'une démo

11.1.1. Le point sur le produit

Toute l'équipe Scrum participe à la réunion. Le Product Owner en est l'animateur principal. Toutes les parties prenantes y sont conviées et leur venue vivement encouragée.

Avec les PP, cela en fait la réunion Scrum à laquelle assistent le plus grand nombre de personnes. La revue est souvent la seule qui intéresse des gens non mobilisés dans Scrum. C'est l'occasion de les faire participer aux décisions sur l'avenir du produit.

11.1.2. Une réunion de fin de sprint

La revue a lieu le dernier jour du sprint, elle sera suivie de la rétrospective. La durée de la revue s'ajuste en fonction de celle du sprint.

Pour un sprint de deux semaines, la revue de l'équipe Peetic dure toujours moins de deux heures et une heure en moyenne.

Dans la pratique, cette limite est assez facile à tenir si on se tient à une démonstration qui ne porte que sur les nouveautés du sprint. La partie **démo** de la réunion va durer environ une vingtaine de minutes.

La revue nécessite une légère préparation, pour être en capacité de faire une belle démonstration. Le temps de préparation est réduit : il n'y a pas de présentation de slides comme habituellement dans les revues de projet.

11.1.3. Une invitation au feedback

La revue porte sur le résultat du travail de l'équipe pendant le sprint.

Dans le cas d'un développement de logiciel, il s'incarne en un incrément incluant les stories finies, accompagné du nécessaire pour le faire fonctionner (documentation, tests, scripts, etc.). C'est cette version opérationnelle qui est présentée en revue ; elle sera ensuite à disposition des parties prenantes sur un environnement dédié.

Le résultat essentiel est une amélioration de la **confiance** entre l'équipe et les parties prenantes. L'équipe est satisfaite d'avoir montré qu'elle a produit un résultat de qualité. Les parties prenantes sont rassurées d'avoir vu des choses fonctionner et d'avoir eu des conversations sur le produit.

11.2. Les activités de la revue de sprint

Voici un enchaînement possible des activités de la réunion :

- Avant la revue, l'équipe **prépare** tout ce qui est nécessaire pour que la démonstration se passe bien.
- Au début de la réunion, le Product Owner **statue par rapport à l'objectif du sprint**.
- Ensuite, pour chaque story finie, l'équipe **effectue une démonstration** et **collecte le feedback** sollicité.
- On profite de la présence des parties prenantes pour **évaluer l'impact obtenu avec le produit** et **prendre une décision sur son avenir**.

11.2.1. Préparer la revue

Logistique

L'équipe s'assure que le matériel nécessaire pour la revue fonctionne bien. La démonstration est généralement faite dans une salle différente de son espace de travail, pouvant accueillir du monde et disposant d'un vidéoprojecteur.

Si des participants sont à distance, la revue passe par l'emploi d'un système de vidéoconférence.

La démonstration s'effectue sur un environnement dédié, le plus proche possible de l'environnement de production. Cela est précisé dans la définition de fini.

Les nouveaux outils de développement permettent de déployer rapidement sur un environnement dit de pré-production, idéal pour la démonstration et pour l'usage ultérieur par les parties prenantes.

Répétition

La préparation de la réunion consiste d'abord à imaginer un scénario d'enchaînement des différentes stories qui seront montrées, avec des jeux de données associés. L'équipe se met d'accord sur le déroulement de la démonstration et sur les personnes qui vont s'exprimer, afin de rendre la présentation fluide et dynamique.

Une répétition est le meilleur exercice pour préparer la démonstration, elle permet de prendre en compte le feedback de l'équipe.

Seules les stories finies seront présentées, mais non celles moitié finies ou presque finies, c'est l'occasion de statuer sur les dernières incertitudes

Stories techniques

Les stories techniques sont destinées à l'équipe, elles n'intéressent pas les parties prenantes. C'est donc au sein de l'équipe qu'elles seront revues.

Après leur examen, l'équipe peut estimer qu'un résumé de ces travaux techniques sera utile lors de la présentation aux parties prenantes.

Ces travaux de préparation peuvent être listés dans la définition de fini pour un sprint et faire l'objet, avec d'autres, de la story de finition de sprint.

11.2.2. Statuer sur l'objectif du sprint

La revue, en présence des parties prenantes, commence à l'heure prévue. La ponctualité est une marque de respect pour les personnes qui sont venues, et le premier pas vers la confiance.

Le Product Owner rappelle l'objectif défini lors de la planification au début du sprint. Il annonce clairement s'il est atteint ou pas.

S'il n'est pas atteint, la discussion sur l'impact pour la release aura lieu en fin de réunion, et la recherche des axes d'amélioration sera conduite lors de la rétrospective.

Il donne la liste des stories qui vont effectivement être montrées.

Pour cela, il peut utiliser les Post-it® des stories placées dans le bac d'arrivée pendant le sprint.

11.2.3. Effectuer la démonstration

L'équipe présente l'incrément réalisé en faisant une démonstration des stories, une par une.

Qui fait la démonstration ?

La démonstration d'une story peut être faite par un développeur qui s'y est particulièrement impliqué. Le risque est le manque d'homogénéité. En outre, certains développeurs ont le clic trop rapide pour un public composé d'utilisateurs. Si l'essaimage a été pratiqué sur une story, les participants à l'essaim sont les responsables de la démonstration de leur story.

Cela peut être aussi le Product Owner ; il est bien placé pour savoir comment s'adresser aux parties prenantes.

Les meilleures démonstrations que j'ai vues étaient faites par un binôme constitué du Product Owner et d'un membre de l'équipe : pendant que le Product Owner présente, l'équipier manipule et complète, éventuellement.

Pour bien marquer le passage d'une story à la suivante, on peut les lister et cocher au fur et à mesure. L'achèvement d'une feature doit aussi être annoncé, après la démonstration de la dernière story permettant de la finir.

11.2.4. Collecter le feedback

Les participants sont invités à s'exprimer sur l'incrément de produit et sur les stories présentées, mais pas sur celles non présentées... Leur feedback se concrétise par des questions, des propositions et demandes de changement. L'équipe répond aux questions. Le bac à sable est enrichi avec les demandes d'évolution suggérées.

Si c'est possible, les personnes présentes sont invitées à manipuler le produit à la fin de la démonstration. Des groupes peuvent se former avec les développeurs qui auront apporté leur ordinateur. Sinon, une version sera à leur disposition après le sprint.

La revue encourage le feedback post-revue.

Pour encourager un retour rapide, on donnera la date limite pour que les propositions soient examinées dans le prochain sprint. Le Product Owner se chargera de le rappeler aux parties prenantes après la revue.

11.2.5. Évaluer l'impact obtenu

La liste des stories et features finies est revue en commun. Le nombre des stories finies (ou de points) est comparé à celui des sprints précédents, et la moyenne calculée. Cette mesure de la vitesse sert à connaître la capacité de l'équipe, utile pour la planification de release.

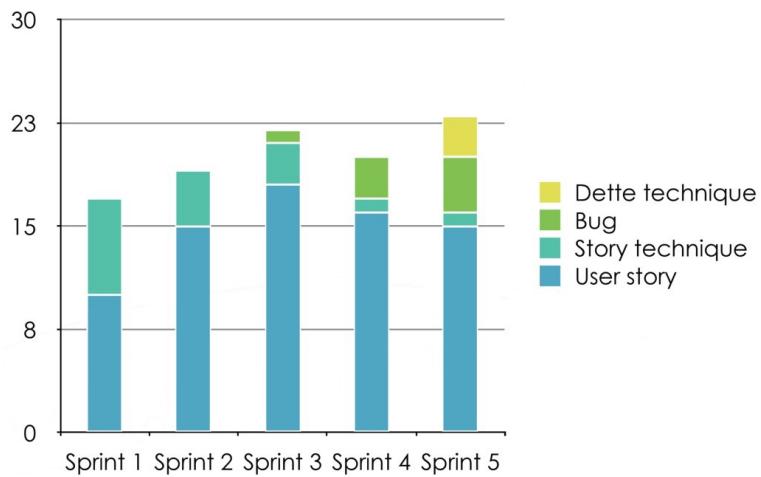


Fig. 11.2 Historique de vitesse.

Avec cet incrément qui s'ajoute aux précédents, on dispose d'un produit qui est utilisable et dont l'impact est évalué pour savoir s'il sera déployé.

Le Product Owner présente la situation, en s'appuyant sur le tableau de features. Les parties prenantes sont invitées à donner leur avis sur l'avenir du produit. C'est l'occasion de discuter avec elles sur le contenu de la release et sur sa date de fin. Des projections peuvent être faites en tenant compte d'hypothèses sur la capacité de l'équipe et sur les impacts attendus.

11.2.6. Décider de l'avenir du produit

C'est le moment de prendre une décision sur le déploiement et sur la date de fin de la release, ou de revoir son contenu, dans le cas d'une release à date fixe. Nous verrons comment le plan de release peut y aider.

Le PO présente également son projet d'objectif pour le prochain sprint. Le feedback des parties prenantes servira, lors de la planification du sprint qui va suivre, à l'ajuster avant discussion avec l'équipe.

11.3. La revue sur le terrain

11.3.1. Attention aux modifications de dernière minute

Le logiciel, c'est souple, des modifications peuvent être faites rapidement. À quelques minutes de la revue, si l'équipe découvre un petit défaut, il est tentant de le corriger. Il faut résister à cette envie, c'est trop tard. On pourrait se dire que si ça ne marche pas, ce n'est pas grave. En fait, cela peut l'être : une modification peut entraîner des régressions. Si le

temps ne permet pas de passer des tests pour s'assurer qu'il n'y a pas de régressions, alors il ne faut pas modifier le code.

J'ai entendu de nombreuses équipes, et pas seulement des équipes d'étudiants, qui, voyant leur démo *planter*, se sont exclamées, de bonne foi : « *mais ça marchait tout à l'heure !* ».

Pour ne pas risquer le fameux « effet démo », il est conseillé de ne pas toucher au code au dernier moment.

11.3.2. Ne montrer que ce qui est fini !

Quand c'est l'équipe qui présente, et en particulier quand c'est un développeur qui fait la démonstration, la tentation est forte de montrer tout le travail réalisé, même si ce n'est pas fini.



Fig. 11.3 On ne montre que ce qui est terminé.

Il faut considérer la démonstration comme un résumé du travail du sprint, dans lequel on ne montre que les actions abouties, de la même façon qu'un résumé de match de rugby ne porte que sur les essais marqués.

11.3.3. Garder les discussions pour la rétrospective

Ce qui compte pour les parties prenantes, c'est le produit partiel avec les stories qu'il contient. La façon dont le sprint s'est déroulé, les tâches et les obstacles, cela ne concerne que l'équipe. Ce n'est pas l'objet de la revue, mais de la rétrospective.

Certaines équipes donnent trop d'importance à un indicateur de sprint (*burndown chart*) et le présentent lors de la revue. Ce n'est pas une bonne idée : il est à usage de l'équipe pendant son sprint, seules les stories finies intéressent les invités de la revue. Ce qu'on peut leur montrer, c'est le *burndown chart* de release, mais pas celui de sprint.

11.3.4. Dérouler un scénario

Une démonstration n'est pas une présentation marketing avec des paillettes, ce n'est pas non plus une recette fonctionnelle avec le passage roboratif des tests.

Pour que l'auditoire comprenne bien, la démonstration doit être fluide. Il est utile de préparer un scénario. Ce n'est pas bien difficile, puisqu'on dispose de la liste des stories ; comme elles sont ordonnées dans le sprint, la démo suivra cet ordre. Ce scénario peut être fourni aux utilisateurs du produit partiel après le sprint.

Attention : la revue n'est pas l'endroit où l'on déroule tous les tests d'acceptation. Ils sont passés avant, et ne sont montrées que les stories les ayant passés avec succès.

11.3.5. Inviter largement, mais expliquer que c'est un produit partiel

La revue de sprint est l'occasion de présenter les résultats, et il faut rechercher l'audience la plus large possible afin de développer des relations entre des gens qui ne se voient pas assez. Des invitations sont envoyées à toutes les parties prenantes, dont la liste doit être connue.

Leur présence à la revue rend inutile la tenue d'autres réunions. Les organisations qui fonctionnent d'habitude avec des comités (comité de pilotage, comité de suivi) devraient les supprimer pour ne conserver que les revues de sprint.

La revue permet d'éviter des démonstrations spécifiques pour des personnes importantes qui n'ont pas daigné se déplacer. Ce serait de la perte de temps que d'aller leur faire plus tard alors que la revue est prévue pour ça. Si quelqu'un d'important ne peut pas assister à la revue, ce n'est pas dramatique, la prochaine reviendra vite, au rythme des sprints.

Dans certaines organisations, on rechigne à inviter des personnes extérieures tant que le produit n'est pas suffisamment complet. Il existe la crainte qu'elles soient frustrées parce que le produit ne contient pas tout ce qu'elles attendent. Ce serait une erreur de se priver des bénéfices du feedback. Le mieux est de les sensibiliser à Scrum et de les inviter à la revue.

On peut leur rappeler qu'il s'agit d'un produit partiel et que les autres fonctions seront réalisées ultérieurement. Une bonne façon de procéder est de présenter le plan de release actuel.

11.3.6. Solliciter le feedback ultérieur

Un processus itératif fonctionne avec le feedback. La revue de sprint est l'endroit idéal pour le solliciter. Les personnes présentes peuvent intervenir pendant la revue et poser des questions lors de la démonstration. L'équipe y répond, soit en précisant un point mal compris, soit en expliquant que cela est déjà dans le bac d'affinage et sera fait plus tard, soit en ajoutant une entrée dans le bac à sable.

Les bacs sont mis à jour avec le feedback, qui se concrétise par la création de nouvelles stories dans le bac à sable. Le Product Owner examinera les demandes avant la planification du sprint, pour éventuellement en inclure une. Certaines requêtes très simples (changement d'un libellé, d'une couleur, etc.) seront regroupées en une seule story.

Le feedback pendant la revue reste limité : les personnes ne manipulent pas elles-mêmes en général. C'est pourquoi il faut pousser à utiliser le résultat du sprint après la revue : pour cela, un environnement spécifique peut être mis à disposition des personnes qui souhaitent essayer le produit et faire ainsi du feedback plus complet.

11.3.7. Découper la revue pour s'adresser au bon public

La revue a deux objectifs :

- collecter le feedback,
- prendre une décision sur la vie du produit.

Ces deux objectifs ne visent pas forcément les mêmes personnes. Le feedback sur les user stories montrées est demandé aux futurs utilisateurs, tandis que l'avancement du produit intéresse des parties prenantes impliquées dans le pilotage.

En outre, en fin de sprint il n'y a pas que des user stories qui sont finies : d'autres types de stories à montrer n'intéressent pas des utilisateurs mais sont importantes pour l'équipe.

Pour optimiser la participation, on peut envisager de faire la revue en trois parties :

- une revue interne par l'équipe Scrum uniquement,
- une revue externe avec les utilisateurs et/ou leurs représentants,
- une revue de pilotage.

La revue interne est une extension de l'activité de préparation, incluant :

- une véritable démonstration des user stories, répétition de ce qui sera montré aux utilisateurs dans la revue externe ;
- une revue des stories qui n'intéressent que les développeurs, abordées uniquement lors de cette revue interne, car elles n'apportent pas de valeur aux utilisateurs ;
- l'examen des stories de découverte, et plus globalement des bacs d'affinage et de départ.

Cette façon de faire peut avoir des impacts positifs :

- on obtient une revue externe plus courte, plus fluide et plus efficace, concentrée sur la démo ;
- on recueille le feedback de l'équipe avant même la démo ;
- on revoit non seulement ce qui est fini, mais également ce qui est prêt ;
- les stories techniques sont examinées par ceux à qui elles sont destinées ;
- le PO et le SM ont le temps de préparer les indicateurs destinés à la revue de pilotage.

Enfin, ce découpage offre la possibilité de placer la rétrospective entre la revue interne et la revue externe, ce qui est parfois plus pratique du point de vue de l'organisation.

L'équipe Peetic organise ses revues un vendredi sur deux. La préparation de la revue a lieu le matin de 10 h à 11 h. On y répète la démo en décidant qui manipulera pendant que Céline présentera la story aux parties prenantes. La

revue a lieu à 14 h. Mona et Gilles y assistent régulièrement. Kevin vient chaque fois qu'il peut, il est parfois accompagné de ses meilleurs clients.

Bien commencer

La question à se poser

Est-ce que l'équipe produit bien un incrément potentiellement livrable à la fin du sprint ?

De mauvais signes

Aucune partie prenante n'assiste à la revue.

La revue ne produit aucun feedback.

L'objectif n'est jamais atteint et il n'y a pas d'action d'amélioration.

Par quoi démarrer

En début de release, demander à toutes les parties prenantes d'inscrire les dates de revue de tous les sprints sur leur agenda.

Une lecture pour tous

« *Des mots, des maux ? Démo !* ». Une présentation d'Émilie et Caro qui vous dit tout sur la revue [Franchomme, *Démo*].

À retenir

La revue de sprint est l'occasion de partager les réalisations de l'équipe avec le reste de l'organisation. Elle permet de communiquer un avancement objectif sur la release. La visibilité apportée et le feedback reçu permettent d'augmenter les chances que le produit soit un succès.

Références :

☞ Émilie Franchomme et Caroline Damour-Nobi, *Des mots, des maux ? Démo !*, 2012.

<http://fr.slideshare.net/carolilo/agile-tour-bordeaux-octobre-2012-des-mots-des-maux-demo>

La rétrospective

Dans de grandes entreprises, quand un projet touche à sa fin, le service méthodes ou son équivalent rappelle au chef de projet qu'il faudrait faire un bilan avant que l'équipe se disloque. C'est l'occasion de revenir sur la façon dont s'est déroulé le projet et de dresser une liste des faits marquants.

Malheureusement, le bilan de projet arrive toujours trop tard : le projet est fini. On peut toujours se dire que d'autres projets bénéficieront des résultats du bilan. Peut-être, mais c'est dommage pour les participants de ne pas avoir appris plus tôt des leçons tirées.

C'est pour permettre à une équipe de s'améliorer régulièrement que la rétrospective de sprint existe. Avec Scrum, on n'attend pas la toute fin d'un projet pour faire des bilans, la « rétro » a lieu à chaque sprint.

En poursuivant l'analogie d'un match de rugby pour le sprint, on peut comparer la rétrospective à la discussion « *on refait le match* », mais à laquelle participeraient uniquement les joueurs, dans le but de faire mieux au match suivant.

Nous avons proclamé que Scrum n'était pas un processus. Pourtant la rétrospective, c'est de l'amélioration de processus : ne serait-ce pas contradictoire ? Le processus n'est pas imposé à l'équipe, c'est elle qui le construit régulièrement à partir du cadre proposé par Scrum, en particulier lors des rétrospectives. À partir des expériences tirées sur le sprint courant, l'équipe identifie ce qui marche bien pour elle, ce qui marche moins bien, et trouve collectivement ce qu'il faut modifier à sa façon de travailler – son processus.

12.1. Une pratique d'amélioration continue

Le terme rétrospective est surtout utilisé dans le domaine artistique pour évoquer la présentation de l'œuvre d'un créateur de façon chronologique : on entend que *France 3* propose une rétrospective Hitchcock pendant l'été, ou on lit dans *La Dépêche* que Les Abattoirs exposent du Saura pour une rétrospective. Cela concerne souvent des artistes morts...

La rétrospective de sprint ne se contente pas de revenir en arrière, elle s'appuie sur le passé pour préparer le futur de l'équipe.

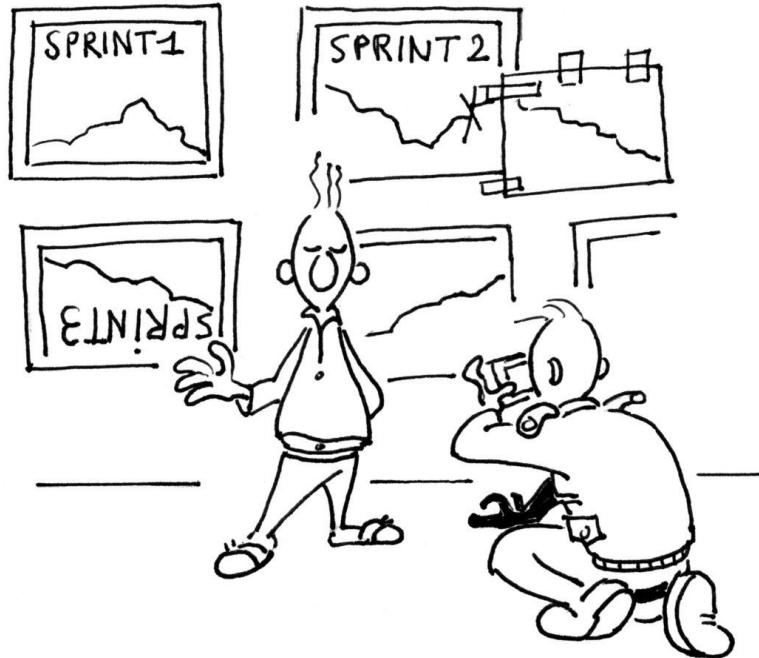


Fig. 12.1 Un ScrumMaster mégalo expose ses burndowns pour la rétrospective de ses sprints

12.1.1. De l'inspection et de l'adaptation

À la fin d'un sprint, la rétrospective est le moment où l'équipe se pose des questions sur la façon dont elle a travaillé. Cela constitue un des piliers de l'approche empirique, l'inspection, conduisant à des adaptations, en fonction du ressenti sur le sprint qui se termine.

Ces inspections et adaptations se poursuivent pendant toute la vie du produit : en effet, il y a toujours des améliorations à faire, il n'existe pas de processus ultime.

12.1.2. Un moment de réflexion collective

La rétrospective constitue un moment particulier où l'équipe s'arrête de produire, prend le temps de réfléchir et parle de ses expériences.

Comme disait le regretté Gébé :

« *On arrête tout, et on réfléchit* »

Une rétrospective permet de capitaliser sur les pratiques qui ont marché, d'éviter de refaire les mêmes erreurs, de partager différents points de vue et de s'adapter aux nouvelles avancées (dans la technologie utilisée pour développer, dans le domaine fonctionnel, dans les pratiques agiles).

D'après mon expérience, une rétrospective dure en moyenne une heure, et a lieu dans la même demi-journée que la revue, vers 16 heures environ.

Après la rétrospective, pour célébrer la fin du sprint, l'équipe Peetic se retrouve au bistrot, c'est la célèbre trilogie : démo, rétro, apéro.

Il est aussi envisageable, pour des sprints courts, de faire en un seul jour la revue et la rétrospective plus la planification du sprint suivant. Dans ce cas la rétrospective aura lieu en fin de matinée (et la trilogie devient démo, rétro, resto...) ou en début d'après-midi.

Pour que la réflexion collective soit bienveillante, il faut partir du principe que tout le monde a fait de son mieux pendant le sprint.

12.1.3. C'est l'équipe qui refait le match

Toute l'équipe Scrum participe à la réunion. On n'oubliera pas que le Product Owner fait partie de l'équipe.

La rétrospective a lieu juste après la revue, mais les parties prenantes n'y restent pas : la confiance nécessaire au succès d'une rétrospective pourrait en pâtir. Dans le cas d'une transition à l'agilité, l'équipe qui s'occupe de la conduite du changement peut être représentée.

La réunion est animée par le ScrumMaster, tant que l'équipe n'est pas vraiment auto-organisée. Dans les situations difficiles, il est préférable que ce soit une personne extérieure à l'équipe qui joue le rôle de facilitateur de cette réunion.

Une rétrospective représente pour les participants une possibilité d'apprendre comment s'améliorer. L'idée est que ceux qui contribuent au résultat sont les mieux placés pour savoir comment progresser. L'objectif est l'apprentissage collectif, et non la recherche de coupables.

12.2. Les activités de la rétrospective

Voici un enchaînement possible des activités de la réunion.

- L'animateur, en général le ScrumMaster, **s'assure que l'environnement est propice à l'expression de tous.**
- L'équipe commence par **collecter les informations** sur le sprint passé.
- Puis elle **identifie des idées d'amélioration** dans la façon de travailler.
- Elle **sélectionne les améliorations** à mettre en place et définit les actions concrètes.
- Enfin, elle revoit ses valeurs et ses pratiques pour **adapter Scrum** pour le prochain sprint.

12.2.1. Créer un environnement propice à l'expression

Il s'agit de s'assurer que tout le monde se sent en confiance et pourra s'exprimer librement pendant la rétro.

S'il y a des doutes sur le climat de confiance qui entoure la réunion, il est préférable de demander à chacun, à bulletin secret, s'il se sent en capacité de s'exprimer librement. Si l'animateur constate que ce n'est pas le cas, la rétrospective ne peut pas avoir lieu, il faudra trouver des conditions différentes qui rétablissent la confiance.

Personnellement, je n'ai jamais rencontré ce cas. Je pense qu'avec la culture française, on n'a pas de problème pour se dire les choses. Il arrive que je demande aux managers de sortir, en leur expliquant que la rétrospective est l'affaire de l'équipe seule.

Avant de passer à la suite, le ScrumMaster rappelle quelle était l'amélioration décidée lors de la précédente rétrospective. Il fait un bilan objectif de la situation. C'est important de montrer que des choses ont changé pour motiver l'équipe à proposer de nouvelles idées de changement. Si ce n'est pas le cas, la cause doit être analysée.

12.2.2. Collecter les informations sur le sprint passé

Dans une rétrospective, on commence par regarder en arrière (dans le rétroviseur). Il s'agit de regarder le passé, non pour juger le comportement de certains ni pour faire son autocritique, mais dans le but de s'améliorer en tant qu'équipe.

L'approche basique consisterait à poser des questions à chaque participant, de façon un peu similaire à ce qui est fait dans la mêlée :

- Qu'est-ce qui a bien fonctionné ?
- Qu'est-ce qui s'est mal passé ?
- Qu'est-ce que nous pourrions améliorer ?

Cependant, le risque avec cette démarche un peu brutale est de ne pas collecter beaucoup d'informations. Pour dépasser les évidences de solution, il est préférable d'utiliser une approche plus progressive en se remettant dans le contexte du sprint. Avant de chercher à améliorer les pratiques, il convient de collecter le ressenti des participants sur ce qui s'est passé pendant le sprint qui s'achève.

Une technique courante qui facilite la collecte est la présentation chronologique de la vie de l'équipe pendant le sprint, appelée *timeline* : la ligne de vie est tracée avec les événements marquants qui ont jalonné la période, pour constituer une frise historique.

On s'appuie sur la liste des obstacles rencontrés pour se remémorer les péripéties du sprint, en les plaçant dans la chronologie au moment où ils ont été détectés.

Ensuite, les événements sont annotés selon leur perception par les membres de l'équipe : agréables, frustrants, fâcheux...

On peut utiliser des notes adhésives de couleur pour repérer les impressions et profiter de l'effet visuel quand elles sont collées sur la timeline. Le partage des ressentis se fait à voix haute.

Il est important d'avoir des retours sur ce qui a bien marché. C'est en capitalisant sur les réussites collectives que l'équipe se construit en tant qu'entité.

12.2.3. Identifier des idées d'amélioration

Il ne s'agit pas seulement de poser des questions sur le passé, mais aussi d'apporter des réponses concrètes qui seront mises en place dans le prochain sprint.

Les informations collectées dans l'étape précédente sont complétées avec les idées d'amélioration proposées par l'équipe.

Je conseille de commencer par une réflexion individuelle, pendant laquelle chaque participant note ses idées sur des Post-it®.

L'ensemble des Post-it® (avec ceux de la timeline) constitue la matière à partir de laquelle l'équipe va maintenant converger vers l'amélioration pour le prochain sprint. Les informations sont regroupées en catégories, de façon à en obtenir entre cinq et dix. Les catégories correspondent en général à des pratiques agiles et aux outils qui les supportent. Chacune est nommée pour que tout le monde l'identifie clairement.

Ce travail est fait collectivement, ce qui est facilité par l'utilisation de notes adhésives. La technique la plus simple est de réaliser une carte par affinités.

Lors des rétrospectives que j'anime, il m'arrive de demander un regroupement en silence, ce qui permet de dérouler cette étape plus rapidement, les discussions ayant eu lieu lors de l'étape précédente.

Les catégories identifiées sont variées. On trouve des informations sur :

- la responsabilité de l'équipe ou pas,
- la façon d'appliquer les pratiques,
- les outils à installer ou à adapter,
- la faisabilité rapide ou différée.

12.2.4. Sélectionner l'amélioration pour le prochain sprint

Il s'agit de définir sur quelle pratique va porter l'effort d'amélioration. Pour un sprint de deux semaines, il est préférable de n'en sélectionner qu'une seule, éventuellement deux.

La technique de sélection doit permettre la participation de toute l'équipe, avec par exemple un système de votes.

Chaque participant de l'équipe Peetic dispose de cinq points à répartir sur les catégories de son choix. L'intérêt est une prise de décision rapide.

Quelle que soit la technique utilisée pour choisir, l'objectif est de renforcer l'implication de chacun et de motiver tout le monde pour mettre en œuvre l'amélioration.

Pour la pratique choisie en vue de l'améliorer, il convient maintenant de définir les actions concrètes qui vont être menées lors du prochain sprint. Les actions sont identifiées par l'équipe, ainsi que par les personnes qui vont les mettre en œuvre, si ce n'est pas toute l'équipe. On pensera à formuler l'impact attendu de l'amélioration, si possible avec un objectif mesurable.

12.2.5. Adapter Scrum pour le sprint suivant

À la fin de la rétrospective, c'est le moment de revenir sur l'usage qui est fait de Scrum.

Le ScrumMaster rappelle quelles sont les valeurs de l'équipe, afin de lancer un débat sur la perception de l'adhésion à ces valeurs. Les dérives sont abordées et des actions décidées pour éviter qu'elles se reproduisent.

L'équipe revoit sa définition de fini et de prêt : elle étudie chaque élément de la liste, décide de le conserver ou pas pour le prochain sprint, change leur niveau et en ajoute éventuellement.

12.3. Les résultats de la rétrospective

12.3.1. Équipe plus soudée

Le résultat essentiel de la rétrospective est qu'elle contribue à avoir une équipe plus soudée, un collectif plus huilé. On a mis en évidence des aspects positifs dans le travail, on a formulé les choses qui n'allait pas, cela améliore l'équipe.

Le rappel des valeurs communes permet de conforter une vision d'équipe partagée.

Il arrive aussi que remontent de la rétrospective des améliorations dont la réalisation se situe en dehors du pouvoir de l'équipe. Elles concernent le management, ou les parties prenantes ou d'autres équipes. Ce sera au ScrumMaster d'être le porte-parole de l'équipe pour faire avancer ces demandes.

L'équipe peut également évaluer son niveau de satisfaction, qui est une mesure primordiale de la santé de l'équipe.

12.3.2. Pratiques améliorées

L'amélioration peut porter sur des pratiques déjà mises en place : limiter la mêlée à 15 minutes, faire une heure de travail en binôme tous les jours, modifier la façon de faire du management visuel sur le tableau, etc.

Les actions concrètes qui en découlent sont décidées immédiatement.

Exemple : la durée des mêlées est l'amélioration choisie par l'équipe Peetic. L'impact attendu est de passer de 20 minutes à 15 minutes. Les actions identifiées sont : apporter un minuteur, annoncer le temps écoulé, afficher la durée de la réunion tous les jours. Emilie propose d'apporter son minuteur « tomate ».

Les actions relatives aux outils nécessitent généralement des études d'impact avant généralisation éventuelle. Par exemple, si l'équipe décide de mettre en place un outil de test automatique, elle ne peut pas le faire dès le prochain sprint. Il faut au préalable étudier plusieurs outils, en choisir un, former l'équipe, etc. L'étude devient une story technique, prioritaire pour le sprint à venir.

Des actions visant à stopper l'accumulation de la dette technique, comme augmenter la couverture de tests, sont ajoutées dans la définition de fini.

En ce qui concerne les idées d'amélioration identifiées et non sélectionnées pour le prochain sprint, le plus simple, si on a procédé avec des Post-it®, est de prendre une photo et de jeter les Post-it®.

En résumé, on peut retrouver des résultats de la rétrospective dans :

- la définition de fini,
- le bac à sable,
- le tableau de l'équipe,
- l'album photo de l'équipe.

Attention : les améliorations décidées ne doivent pas remettre en question les fondements de Scrum.

12.4. La rétrospective sur le terrain

12.4.1. Attention à ne pas tomber dans la routine

Pour éviter de tomber dans la routine, il est essentiel de varier la façon de les mener. Il existe beaucoup de techniques différentes pour les rétrospectives [Baldauf, Retromat] et il s'en crée continuellement de nouvelles [Pernot, Festival].

En voici quelques-unes parmi celles que j'utilise le plus fréquemment :

- l'étoile de mer,
- la rétrochâtaigne,
- le bateau rapide (*Speed boat*, un des *innovation games*).

12.4.2. Et si rien ne bouge ?

Il n'y a rien de plus frustrant pour les participants à une rétrospective que de constater, sprint après sprint, que rien ne bouge vraiment. C'est le rôle du ScrumMaster de s'assurer que les actions décidées sont vraiment mises en œuvre.

La rétrospective de sprint revient régulièrement : à chaque fin de sprint. Il ne sert à rien d'être trop ambitieux en cherchant à adapter la façon de travailler sur des nombreuses pratiques.

Vouloir faire un trop grand nombre d'actions conduit au risque qu'aucune n'aboutisse vraiment. Pour des sprints courts, d'une semaine ou deux, il vaut mieux essayer d'améliorer une seule chose à la fois. Pour des sprints plus longs, on se limitera à deux.

On peut se dire que les améliorations qui ne sont pas choisies lors d'une rétrospective ont de bonnes chances de l'être dans une prochaine.

Parfois, les équipes se lassent des rétrospectives, estimant qu'il n'y a plus rien à améliorer, ou que « *nous disons toujours la même chose* ». L'erreur la plus grave est d'arrêter de faire des rétrospectives. Ne pas utiliser cette possibilité de feedback sur la façon de travailler, c'est passer à côté d'un des bénéfices majeurs de Scrum.

Il y a toujours des choses à améliorer. Par exemple, une idée d'amélioration sortant du cadre traditionnel serait une demi-journée par semaine pour travailler sur des projets personnels.

12.4.3. Ce n'est pas une séance de règlement de comptes

La rétrospective permet à toute l'équipe de s'exprimer et chacun est invité à participer à l'identification des dysfonctionnements constatés pendant le sprint. La mise en évidence d'erreurs ne doit pas conduire à accuser une personne d'en être responsable.

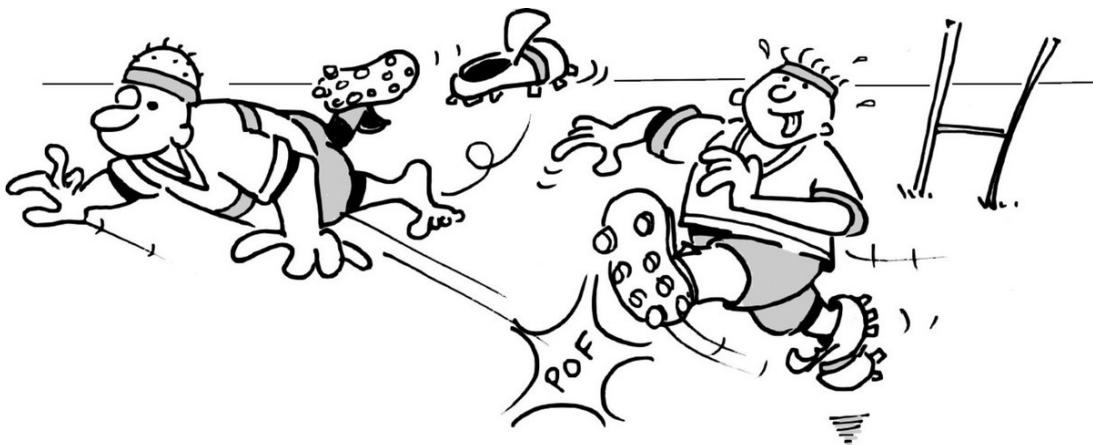


Fig. 12.2 La rétrospective ne doit pas tourner au règlement de comptes

Le ScrumMaster doit veiller à ce que la rétrospective ne dérive pas vers les attaques personnelles, en faisant de l'équipe l'objet central des discussions.

Il arrive aussi que des dysfonctionnements soient provoqués par des entités extérieures. C'est fréquent : l'équipe n'est pas dans sa bulle, elle n'est jamais complètement indépendante.

Il est difficile de définir des actions dont les responsables ne participent pas à la réunion. Le ScrumMaster doit remonter les problèmes à la hiérarchie et tout faire pour qu'ils soient traités. Cependant, l'objectif de la rétrospective est d'abord de se concentrer sur son processus et d'identifier des actions que l'équipe peut mettre en œuvre elle-même.

12.4.4. Parler de ce qui va bien

Comme l'objectif est d'améliorer des façons de faire, la collecte passe essentiellement par la mise en évidence de pratiques qui ne se passent pas bien, en tout cas pas aussi bien qu'elles le pourraient.

Pour équilibrer ce point de vue qui peut être perçu comme négatif, il convient aussi de mettre en exergue ce qui s'est bien passé pendant le sprint, les pratiques qui ont apporté de la satisfaction à l'équipe.

Cela contribue à entretenir un climat positif et constructif, et aussi à progresser en rappelant ce qui a bien fonctionné, jusqu'à ce que cela devienne une habitude.

12.4.5. Mener des rétrospectives plus poussées en fin de release

La rétrospective de sprint revient sur le proche passé du sprint qui se finit. À chaque fin de release, il est souhaitable de mener une rétrospective qui porte sur ce qui s'est passé pendant les quelques mois de la release.

À cette occasion, il est conseillé de prévoir une réunion plus longue que d'habitude et d'organiser la réunion différemment.

12.4.6. Utiliser un facilitateur externe

Quand le ScrumMaster n'y arrive pas, quand il y a des blocages qui ne permettent pas à l'équipe de s'exprimer, l'intervention d'un consultant extérieur est souhaitable.

En période de crise, quand ça va mal, que la date de livraison est déjà dépassée par exemple, on cherche à parer au plus urgent, et croyant gagner du temps... on élimine la rétrospective.

C'est dommage, car c'est faire sauter la dernière chance d'éviter le pire. Un œil extérieur peut justement permettre de faciliter l'identification collective des problèmes majeurs de l'équipe.

12.4.7. Varier les objectifs

Une rétrospective de sprint n'a généralement pas d'objectif précis, implicitement on aborde la façon de travailler ensemble. De temps en temps, on peut lui donner un objectif particulier, sur un sujet technique ou sur un principe ou une valeur qui pose problème.

Voici un exemple qui vise à renforcer le principe de rythme soutenable.

La rétro-glandouille

- **Introduction**

Le facilitateur présente l'objectif de la rétrospective : ajuster le niveau de mou dans la vie d'une équipe pour un sprint, dans le but de conserver un rythme soutenable ou d'y revenir.

- **Chacun réfléchit**

L'animateur demande aux participants leurs idées de glandouille sur plusieurs plans : individuel, en petits groupes et pour toute l'équipe. À quoi ont-ils envie de glandouiller pendant le prochain sprint ?

La réflexion est d'abord personnelle : chacun écrit ses idées sur une note adhésive. On prend trois couleurs de notes, une par plan. On demande aux participants d'ajouter la durée de glandouille souhaitée sur la période, quand c'est pertinent. Par exemple : faire une pause à 16 h 00, pour 10 minutes. On attend (environ 5-6 minutes) que tout le monde ait exprimé au moins une idée équipe et une idée groupe, et éventuellement plusieurs idées individuelles.

- **On tourne**

Ensuite, chacun fait passer son idée de groupe à son voisin de droite et son idée d'équipe à son voisin de gauche. On garde les idées de glandouille individuelle.

Le voisin prend connaissance de l'idée de glandouille et donne son avis : il coche le Post-it® s'il aime l'idée ou fait une croix si cela ne lui plaît pas, puis il fait passer au voisin suivant. Il peut aussi être indifférent à l'idée.

On continue jusqu'à avoir fait le tour complet des participants.

- **On collecte**

L'animateur aura préparé un *paperboard* avec trois zones (individuel, groupe, équipe). Chacun vient coller les Post-it® ou bien on inscrit le résumé des fiches en faisant les comptes des adhésions et objections.

- **On décide des améliorations**

On discute des idées qui totalisent le plus de différence positive entre les adhésions et les objections, dans chaque zone. Pour les idées individuelles, on compte simplement celles énoncées plusieurs fois. Cela donne trois ou quatre idées qui peuvent être mises en œuvre pour le prochain sprint.

- **On identifie sur quoi passer moins de temps**

Pour finir, on demande quelle activité de la période passée aurait pu être raccourcie ou supprimée pour introduire cette glandouille. Il faut bien prendre le temps supplémentaire à consacrer à la glandouille sur d'autres activités. Cela permet aussi d'identifier les activités ressenties comme longues ou inutiles.

- **À noter**

La rétro-glandouille suit un déroulement différent des rétros habituelles : au lieu de s'appuyer sur le passé pour préparer l'avenir comme dans les rétros habituelles, on se place dans un futur idéalisé de glandouille pour finir par revenir sur le passé.

Bien commencer

La valeur Scrum

Le respect qui, pour la rétrospective, consiste à penser que chacun a fait de son mieux pendant le sprint.

La question à se poser

Est-ce que les conditions sont favorables pour bien se dire les choses ?

De mauvais signes

L'équipe a l'impression de perdre son temps parce que rien ne bouge.

Tout va bien, l'équipe ne trouve rien à améliorer.

Par quoi démarrer

Bien préparer la première rétro, avec une formule ludique.

Une lecture pour tous

Tirer parti des rétrospectives agiles [Gonçalves, Retrospectives], un petit livre traduit en français.

À retenir

La rétrospective implique toute l'équipe dans une réflexion, en vue de faire émerger des façons de mieux travailler.

L'équipe s'appuie sur le passé pour préparer le futur. Elle collecte les informations marquantes du sprint, puis exprime son ressenti sur ce qui c'est passé. Ensuite, elle découvre des axes d'amélioration, en sélectionne un ou deux et définit son plan d'actions.

La rétrospective est aussi le moment où l'équipe bâtit une culture commune sur ses succès. C'est la porte d'entrée vers l'amélioration continue.

Références :

☞ Corinna Baldauf, *Retromat*, Web.

www.plans-for-retrospectives.com/

☞ Luis Gonçalves et Ben Linders, *Tirer parti des rétrospectives agiles*, VF, 2014.

https://leanpub.com/gettingvalueoutofagileretrospectives_FR/

☞ Pablo Pernot, Festival de rétrospectives, 2015.

<http://www.areyouagile.com/2015/03/festival-de-retrospectives/>

Contextualiser Scrum

Ce chapitre n'est pas à cette place par hasard. Il vient après la présentation de ce qui constitue le cadre Scrum.

Un cadre, ce n'est pas une méthode, encore moins une méthodologie ou un processus complet qu'on pourrait « instancier » pour l'appliquer directement. Pour utiliser Scrum, il faut d'abord compléter ce cadre avec des pratiques complémentaires, variables selon le domaine, et adapter le tout au contexte.

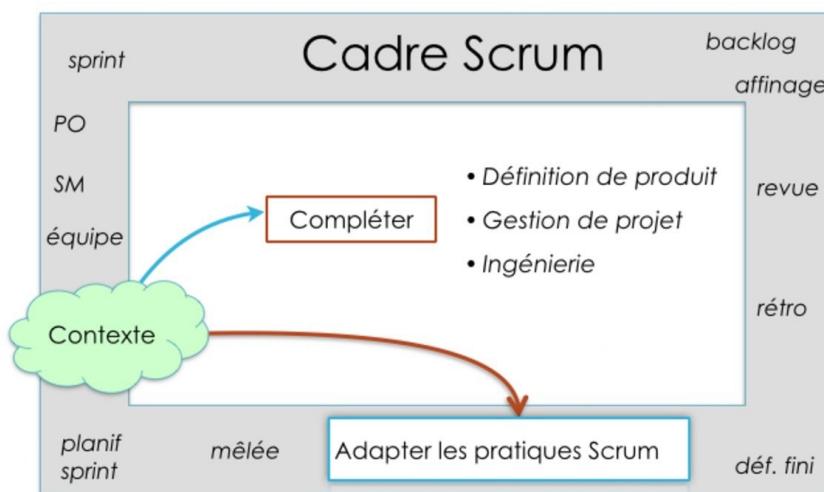


Fig. 13.1 Le cadre Scrum est adapté et complété

Pour les compléments, il est naturel d'aller voir des pratiques issues d'autres disciplines, comme la définition de produit, la gestion de projet et l'ingénierie du domaine concerné.

Dans ce chapitre, nous allons étudier comment caractériser le contexte pour en déduire des pratiques Scrum à adapter et des pratiques « non Scrum » à ajouter.

13.1. Pratiques agiles

La notion de **pratique** est centrale pour l'adaptation de Scrum : à côté des valeurs et des principes agiles, qui sont universels, les pratiques varient avec chaque situation.

Une **pratique** est une approche concrète et éprouvée qui permet de résoudre un ou plusieurs problèmes courants ou d'améliorer la façon de travailler lors d'un développement.

Exemple : la revue de sprint, avec sa démonstration.

13.1.1. Pratiques Scrum

Dans le cadre Scrum, on peut identifier une quinzaine de pratiques, qui correspondent *grossost modo* aux chapitres du livre :

- Les rôles : Product Owner, ScrumMaster, équipe auto-organisée.
- Les événements : sprint, affinage du backlog, planification de sprint, mêlée, revue de sprint, rétrospective.
- Les artefacts : incrément de produit, backlog, liste des obstacles.
- Les définitions de prêt et de fini.

13.1.2. Pratiques complémentaires

Il existe de nombreuses pratiques dans la mouvance agile. Leur nombre, la façon de les nommer et leur classification varient selon les auteurs. Il n'y a pas de liste officielle.

Jurgen Appelo avait tenté une synthèse sur son blog, « *The Big List of Agile Practices* »^[1], élaborée à partir de huit sources différentes.

En France, l'Institut Agile a publié une liste d'une soixantaine de pratiques agiles [Institut, Référentiel] et a produit une carte des pratiques, qui donne une idée de leur origine^[2].

La façon d'organiser ces pratiques est très variable. La classification peut s'opérer selon différentes caractéristiques :

- l'origine de la pratique, pour préciser de quelle approche elle provient (Scrum, XP, Kanban, etc.) ;
- les activités ou disciplines du développement (conception, codage, gestion de projet, test, etc.) ;
- le fait qu'elle soit obligatoire ou optionnelle.

La nature optionnelle d'une pratique est sujette à discussion. Comme Scrum n'est qu'un cadre, c'est plus facile : mon opinion est que les pratiques Scrum sont obligatoires et que c'est la façon de les mettre en œuvre qui diffère suivant le contexte.

Les pratiques venant régulièrement en complément à Scrum sont présentées en trois catégories :

Pour la définition de produit

Quand on utilise Scrum pour développer un nouveau produit, il est souhaitable d'y adjoindre des pratiques facilitant la constitution initiale du backlog. Dans les chapitres suivants, nous aborderons les pratiques *impact mapping*, *story mapping*, *user story* et tests d'acceptation.

Pour la gestion de projet

Scrum est souvent perçu comme une méthode de gestion de projet : on oppose son approche empirique à l'approche prédictive. Cependant, des pratiques de management complémentaires sont utiles selon les situations. Quelques-unes seront présentées dans les

chapitres suivants : *Planifier la release*, *Appliquer Kanban à Scrum*, *Améliorer la visibilité avec des indicateurs*, *Transformer les organisations*.

Pour le développement de logiciel

Encore aujourd’hui, la plupart des mises en œuvre de Scrum ont comme objectif de développer du logiciel. Mais Scrum ne prescrit pas de technique d’ingénierie particulière pour le développement, laissant le choix à l’équipe. C’est la différence majeure avec XP, qui propose des pratiques dédiées au développement de logiciel.

Nous verrons que la plupart sont nécessaires et induites par la signification de fini : intégration continue, pilotage par les tests, conception émergente, etc.

13.2. Caractériser le contexte

L’idée est que le contexte d’un développement avec Scrum peut être défini avec quelques attributs.

13.2.1. Influence de l’organisation

Un développement s’inscrit dans le contexte d’une organisation. Une organisation possède une culture, un savoir-faire, bref des conditions qui ont une influence sur les projets qui vont appliquer Scrum et l’agilité. Il est bien évident que l’application sera plus difficile dans une entreprise qui a des valeurs et des principes éloignés de ceux définis par le *Manifeste agile*.

Nous reviendrons sur l’influence d’une organisation dans le chapitre 22 *Transformer les organisations*.

Les projets menés dans une organisation sont influencés, mais pas forcément tous de la même façon, c’est pourquoi il convient avant tout de définir le contexte du projet qui va utiliser Scrum.

13.2.2. Contexte du projet

Le terme projet est utilisé ici par habitude et facilité, mais ne figure pas, comme nous l’avons vu précédemment, dans le vocabulaire Scrum.

Les attributs de contexte permettent de caractériser un projet, nous en présentons dix significatifs.

Cette notion de contexte s’appuie sur l’**agilité en situation**^[3], et les travaux de Philippe Kruchten [Kruchten, *Context*].

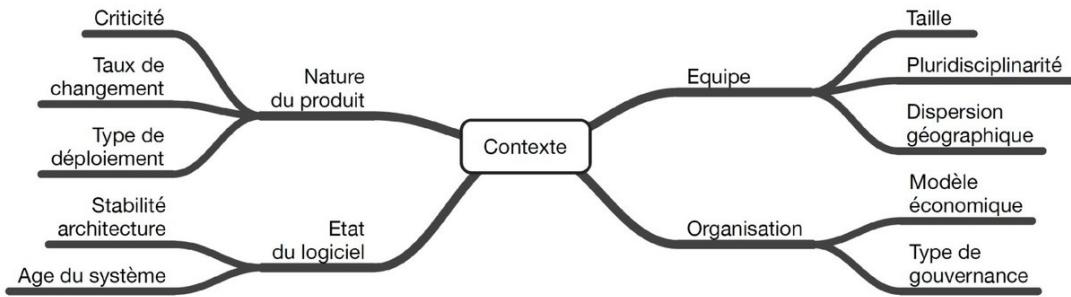


Fig. 13.2 Les attributs définissant le contexte d'un projet

- **Taux de changement** : la fréquence des changements sur le produit, à propos de sa technologie ou des demandes des utilisateurs.
- **Criticité** : l'enjeu humain ou économique en cas de défaillance du logiciel.
- **Déploiement** : le nombre d'instances du produit final et la façon de les déployer, ainsi que la fréquence de déploiement.
- **Taille** : le nombre de personnes dans l'équipe. Elle est représentative de la taille du projet, comme le sont aussi le budget ou la taille du code.
- **Pluridisciplinarité** : la capacité des membres de l'équipe dans les pratiques d'ingénierie, combinaison de leur formation et de leur expérience.
- **Dispersion géographique** : le nombre de bureaux différents accueillant les personnes de l'équipe.
- **Âge du système** : la quantité de code déjà existant qu'incorpore le logiciel à développer, et sa qualité.
- **Stabilité de l'architecture** : le degré selon lequel on peut ajouter des *user stories* au logiciel existant sans revoir l'architecture.
- **Modèle économique** : projet Open Source, projet interne, projet sous-traité au forfait, logiciel commercial, etc.
- **Type de gouvernance** : les contraintes imposées par l'organisation sur le cycle de vie et le « reporting » des projets.

13.3. Étudier l'impact sur les pratiques

Le sujet de ce paragraphe n'est pas l'éligibilité de Scrum selon les caractéristiques des projets, c'est l'adaptation des pratiques aux contraintes. Il permet de se rendre compte des efforts nécessaires pour contextualiser Scrum. Mais ce n'est pas parce qu'une situation demande beaucoup d'efforts que l'usage de Scrum n'y est pas conseillé : il faut prendre en compte les bénéfices, qui peuvent être encore plus substantiels.

Avant d'adapter, il faut d'abord aligner les valeurs de l'organisation avec celles de l'agilité. Scrum est avant tout un système de valeurs et un état d'esprit. Rien ne sert d'adapter des pratiques sans adhésion aux valeurs et sans partage de l'esprit agile.

Adapter fait référence aux pratiques présentées dans les chapitres 1 à 12 ; compléter porte sur des pratiques qui seront présentées dans les chapitres 14 à 22.

13.3.1. Nature du produit

Ces attributs sont relatifs au domaine métier dans lequel se situe le produit développé.

Tab. 13.1 Impacts liés à la nature du produit

Attribut	Mesure significative	Impact sur les pratiques (A : adaptation de pratiques Scrum, C : complément)
Taux de changement	Élevé (urgences)	A : Story de réserve pour les urgences. C : Kanban appliqué à Scrum.
Criticité	Forte	A : Affinage et réalisation nécessitant la présence importante d'experts des normes à suivre. C : Pratiques poussées de définition de produit (traçabilité) et de développement de logiciel (preuve).
Déploiement	Une seule instance (web)	A : Déploiement (et éventuellement revue) à un rythme désynchronisé des sprints et des releases. C : Déploiement continu incluant des tests de non-régression, Personas.
	Très nombreuses instances	A : Définition de fini évolutive, storyotypes. C : Plan de release pour la synchronisation avec d'autres entités impliquées dans la chaîne de valeur.

Taux de changement

La possibilité de changer pendant le développement est souvent le facteur qui va déclencher le passage à l'agilité. Dans l'esprit des gens, on trouve cette idée de l'agilité : « *on ne sait pas vraiment ce qu'on veut et on sait que ça va changer, alors comme l'approche traditionnelle ne marche pas, essayons une méthode agile* ».

Scrum est effectivement ouvert au changement, mais l'équipe est protégée pendant le sprint. Personne ne peut lui ajouter du travail ou changer l'objectif sans son accord.

Dans de nombreuses organisations, et pas seulement des petites, il est difficile de respecter cette règle, à cause d'une habitude de travail dans l'urgence. Or Scrum pousse à différer les urgences au prochain sprint pour ne pas perturber l'équipe. Ce n'est pas toujours possible, et cela peut représenter un changement radical pour certaines organisations.

Un taux de changement est élevé quand l'objectif du sprint n'est pas souvent atteint ou change pendant le sprint.

Criticité

La criticité est forte quand des vies humaines ou des impacts économiques importants sont en jeu. Lorsque le produit développé possède une criticité élevée, la conformité à des

normes doit être prouvée, et des audits externes ont lieu, portant souvent sur des documents.

Déploiement

Le nombre d'instances déployées varie entre un et des millions selon les applications.

Déploiement unique : dans le cas de l'hébergement d'une seule instance en ligne (cloud), le déploiement est maîtrisé par l'organisation, qui peut le faire très fréquemment.

Déploiement en masse : à l'inverse, un déploiement du logiciel sur des dispositifs diffusés à des milliers d'exemplaires, comme dans le domaine de la téléphonie ou celui des jeux vidéo, rend les mises à jour difficiles et coûteuses, ce qui oblige à avoir des tests très poussés en interne ; l'équipe Scrum ne sera généralement pas impliquée jusqu'au déploiement, ce qui ajoutera des dépendances et des retours tardifs.

13.3.2. Équipe

Ces attributs dépendent de l'équipe qui développe le produit.

Tab. 13.2 Impacts liés à l'équipe

Attribut	Mesure significative	Impact sur les pratiques (A : adaptation de pratiques Scrum, C : complément)
Taille	Plusieurs équipes	A et C : Il faut plus d'une équipe et donc passer au Scrum à l'échelle, décrit dans un chapitre dédié.
	Petite équipe	A : Le PO ne joue pas son rôle à plein temps, le SM est en même temps développeur.
Pluri-disciplinarité	Équipe non pluridisciplinaire	A : Dépendance forte sur des experts, à contrôler et réduire dans la définition de prêt et l'affinage. C : Binôme pour acquérir des compétences.
Dispersion géographique	Un ou deux dév. en télétravail	A : La mêlée est adaptée à l'éloignement. Les développeurs en télétravail se déplacent pour la revue et la rétrospective. À définir pour l'affinage.
	Dispersion, même culture	A : Les événements du sprint se font avec des outils permettant de communiquer à distance. C : Rencontre physique à l'occasion de la fin de release.
	Dispersion, culture différente	Il vaut mieux éviter de constituer une équipe Scrum qui soit dans ce cas, en regroupant les gens différemment pour former une équipe.

Taille du projet

La taille idéale d'une équipe est de cinq à neuf personnes. Pourtant, une enquête que j'ai menée en juin 2013 montre que 10 % des équipes ont une taille de une à trois personnes et presque autant ont dix personnes ou plus.

Plusieurs équipes : à plus de dix jusqu'à des centaines, il faut donc constituer plusieurs équipes Scrum.

Petite équipe : en ce qui concerne les équipes réduites, à quatre ou moins, des retours d'expérience montrent qu'il est possible d'appliquer Scrum.

Pluridisciplinarité de l'équipe

Une équipe Scrum doit couvrir, avec l'ensemble de ses membres, toutes les activités nécessaires pour obtenir un produit à la fin d'un sprint. Si ce n'est pas le cas, elle n'est pas pluridisciplinaire.

Dispersion géographique

L'idéal est que l'équipe soit regroupée dans le même espace. La distribution géographique rend toutes les pratiques plus difficiles à appliquer et plus susceptibles d'échouer. En particulier, la communication entre les personnes, mise au cœur de l'agilité, devient plus compliquée. L'équipe doit trouver une solution acceptable pour la mêlée : vidéoconférence, audio, photo, outil collaboratif, etc.

Un ou deux développeurs en télétravail : c'est le cas d'une équipe avec une ou deux personnes en télétravail – on dit en « *remote* ».

Exemple Peetic :

José, un des développeurs de Peetic, travaille de chez lui dans l'Aveyron, alors que l'équipe est à Toulouse. La pratique de la mêlée, parmi d'autres, est impactée. Le choix fait pour s'adapter est que le ScrumMaster sera avec José au téléphone pendant la réunion et qu'à la fin, il lui enverra la photo du tableau actualisé.

Pour la revue, la rétrospective et la planification du sprint suivant, José viendra à Toulouse pour y participer physiquement. Pour faciliter son déplacement, il est décidé de faire les trois dans la même journée : la revue à 10 heures, la « rétro » à 14 heures et la « planif » à 15 h 30.

Dispersion, même culture : l'équipe est répartie, éventuellement dans plusieurs pays, mais les membres partagent la même langue et se rencontrent régulièrement.

Dispersion, culture différente : l'équipe est divisée, avec des membres dans plusieurs pays à culture différente, qui ne se connaissent pas toujours.

13.3.3. État du logiciel

Ces attributs sont relatifs à l'état du logiciel sur lequel travaille l'équipe.

Tab. 13.3 Impacts liés à l'état du logiciel

Attribut	Mesure significative	Impact sur les pratiques (A : adaptation de pratiques Scrum, C : complément)

Architecture	Pas encore stabilisée	A : Affinage à adapter pour prendre en compte les risques techniques (priorité et estimation délicates), définition de fini pour des stories techniques, revue. C : Décomposition de story.
Âge du système	Présence de « Legacy code »	A : Affinage à adapter pour prendre en compte le remboursement de la dette technique. C : Pratiques poussées de développement de logiciel pour réduire la dette technique.

Architecture

Si l'architecture d'un logiciel n'est pas stable, cela veut dire qu'il y a des risques techniques, qu'il faut réduire au plus tôt. Or il est plus facile de développer et finir des *user stories* simples que de consolider l'architecture avec des stories significatives ce point de vue.

Age du système

Il est plus facile de mettre en œuvre Scrum sur un logiciel nouveau. Si le produit incorpore des parties de code existant (*legacy*), il faut vivre avec ce code, qui est de plus ou moins bonne qualité, probablement avec de la dette technique.

13.3.4. Contraintes de l'organisation

Ces attributs sont caractéristiques de l'organisation qui englobe l'équipe Scrum.

Attribut	Mesure significative	Impact sur les pratiques (A : adaptation de pratiques Scrum, C : complément)
Modèle économique	Contrat client fournisseur	<p>A : Le backlog, car il faut définir sa relation avec la spécification associée à l'appel d'offres. Le moment où commence le contrat est à préciser. Le rôle de Product Owner devrait être côté donneur d'ordre et intégré à l'équipe, ce qui est souvent difficile et peut conduire à une adaptation risquée mais nécessaire pour le bon déroulement du contrat : le PO délégué (côté fournisseur).</p> <p>C : Définition de produit, planification de release avec suivi du budget, estimation.</p>
	Open Source	<p>A : L'équipe n'étant pas à plein temps, la notion de sprint est délicate.</p> <p>C : Pratiques d'ingénierie.</p>
Gouvernance	Forte et traditionnelle	<p>A : Cycle des sprints, cohabitation de Scrum avec les phases et les jalons de l'organisation.</p> <p>A : Définition poussée de l'implication des parties prenantes, qui doivent être sensibilisées à Scrum, revue faisant office de comité de pilotage.</p> <p>C : Indicateurs à expliquer, transition à Scrum et capitalisation au niveau de l'organisation.</p>

Fig. 13.3 Impacts liés à l'organisation

Modèle économique

Il s'agit de la façon dont le budget pour le développement est obtenu. Plusieurs contextes sont possibles, notamment :

Contrat client fournisseur : un développement réalisé par un contrat au forfait entre un donneur d'ordre et une SSII, dont on se doute qu'il complique la mise en œuvre de plusieurs pratiques.

Open Source : les contributeurs travaillant sur les projets open source sont souvent répartis sur plusieurs pays et participent à temps partiel.

Le développement par une équipe interne constitue la situation demandant le moins d'adaptation. C'est le cas des éditeurs.

Gouvernance

Une gouvernance légère ne demandera pas d'adaptation particulière.

Gouvernance forte et hiérarchique : en France, quelques administrations et grandes entreprises ont une façon de gouverner les projets qui est souvent héritée de méthodes mises au point dans les années 1980 dans le domaine industriel. Bien sûr, chacune a ensuite fait ses adaptations à ces méthodes, mais le résultat en matière de gouvernance ne donne pas, en général, dans la légèreté.

13.4. Adapter en fonction de la situation

13.4.1. Situation d'un projet

L'examen des attributs pour un projet permet de définir les pratiques à adapter.

Un projet typique pour un passage à Scrum « facile » aurait les attributs suivants :

- Une équipe de sept personnes, toutes dans le même bureau.
- Un logiciel nouveau avec une architecture connue.
- Un produit qui n'est pas critique et se déploie facilement.
- Un développement fait en interne avec, dans l'équipe, de bonnes compétences techniques.
- Une gouvernance accommodante.

C'est relatif ! Un changement de culture, comme l'est un passage à Scrum, demande toujours des efforts.

Quelques projets se situent dans ce cœur de cible. Mais il en existe beaucoup d'autres qui sont dans une situation demandant plus d'efforts lors de l'introduction de Scrum. Une représentation synthétique donne une idée de l'effort nécessaire pour l'adaptation.

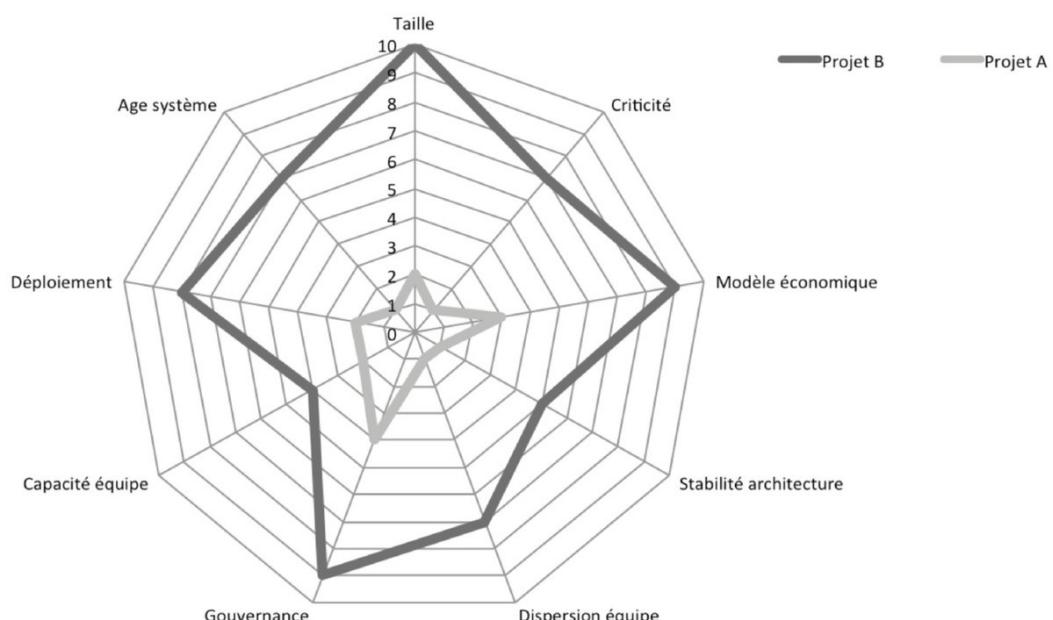


Fig. 13.4 Contexte de projet

Le projet B (plus loin du centre) demande plus d'efforts pour appliquer Scrum que le projet A (au centre), ce qui ne veut pas dire qu'il ne faut pas passer à Scrum pour B : les

bénéfices seront peut-être bien plus importants que le coût de la transition.

13.4.2. Scrum contextualisé

Après avoir analysé le contexte, on connaît les contraintes imposées à la mise en œuvre de Scrum. Pour chaque attribut dont la mesure est significative, il convient de définir comment adapter les pratiques impactées.

Le ScrumMaster anime ce travail collectif de contextualisation au cours du sprint zéro et fait en sorte qu'il soit appliqué, puis ajusté.

Le résultat de cette contextualisation n'est pas consigné dans un document. Le partage de connaissance entre les membres de l'équipe se pratique de façon continue, à travers les événements que nous avons présentés dans les chapitres précédents ; il rend inutile l'écriture d'un document, l'équipe n'en a pas besoin.

En revanche, sa communication à l'extérieur, au niveau d'une organisation qui possède plusieurs équipes et souhaite capitaliser, peut demander plus de formalisme.

13.5. La contextualisation sur le terrain

13.5.1. Attention à ne pas oublier les pratiques d'ingénierie

Une équipe qui se contente du cadre Scrum, sans mettre en place de pratiques d'ingénierie, va se heurter rapidement à des difficultés.

Scrum, avec la définition de fini notamment, permet de les révéler, parfois brutalement – c'est une de ses forces – mais c'est à l'équipe de faire les efforts pour éviter l'accumulation de dette technique.

13.5.2. Décontextualiser

En appliquant une pratique strictement, comme on l'a lu dans les livres ou à partir d'une expérience qui a été racontée, on risque de se couper de l'environnement réel de l'organisation. Ce n'est pas parce qu'une pratique a fonctionné pour un projet qu'elle marchera à l'identique dans un contexte différent.

Un exemple est raconté par Henrik Kniberg. Dans son livre *Scrum et XP depuis les tranchées*, il avait présenté une technique appelée *focus factor*. De nombreuses équipes ayant lu le livre au moment de leur passage à Scrum l'ont appliquée, croyant bien faire. Kniberg l'a constaté avec regret, expliquant que cet usage du *focus factor* était contextuel et qu'il ne l'avait pas utilisé dans d'autres situations.



Fig. 13.5 Attention à la dé-contextualisation : tout content d'avoir reçu un beau livre, on pourrait être tenté de suivre aveuglément ce qui y est écrit

Comme nous le voyons dans ce chapitre, il n'existe pas de Scrum « académique », seul le cadre est défini : tous les projets sont différents et constituent un contexte spécifique pour la mise en place des pratiques.

13.5.3. Adapter sans discernement

Certaines équipes pensent appliquer Scrum, mais n'utilisent que des bribes de l'ensemble des pratiques. Partant du principe que Scrum s'adapte, elles n'en prennent que ce qui les arrange ou uniquement ce qu'elles ont compris d'un apprentissage trop rapide.

S'il y a bien des pratiques optionnelles dans l'agilité, le cadre Scrum forme un tout cohérent duquel il n'est pas opportun d'enlever des parties. Il est plutôt conseillé de chercher les raisons des difficultés rencontrées.

Je vois des équipes qui ont démarré Scrum depuis quelques mois, sans formation ni assistance, et n'ont conservé que la mêlée et des sprints. Évidemment, les résultats obtenus seront décevants.

Nous avons déjà évoqué le « Shu Ha Ri », et nous pouvons résumer ainsi la première phase, le « Shu » :

« *L'équipe suit les pratiques Scrum, elle ne peut pas en enlever, le cadre Scrum n'étant pas négociable ; elle doit les adapter au contexte.* »

13.5.4. Réexaminer régulièrement la situation

La plupart des pratiques agiles sont utiles pour la plupart des projets, mais elles ne s'appliquent pas partout de la même façon, et leur application évolue dans le temps.

Les attributs définissant le contexte changent avec le temps. Après quelques mois, certains devraient se rapprocher du centre du graphe, comme la gouvernance, mais d'autres

peuvent s'en éloigner, comme la taille de l'équipe si le projet grandit.

C'est pourquoi l'analyse du contexte devrait être mise à jour à chaque nouvelle release. La comparaison des contextes sur deux releases successives permet de voir ce qui a évolué et d'en tenir compte dans l'adaptation.

13.5.5. Doser ses efforts en fonction des objectifs

Les résultats obtenus dans l'utilisation de Scrum sont fonction des efforts consentis pour cette contextualisation, et de l'expérience acquise.

C'est ce que nous montre le modèle « Agile Fluency » [Shore, Fluency]. Il utilise la perception de la valeur « métier » produite par une équipe pour situer sa « maîtrise de l'agilité », en termes d'étoiles.

Pour le résumer rapidement, l'utilisation basique de Scrum mène à une étoile, la mise en œuvre de pratiques d'ingénierie avec une définition de fini complète fait passer à deux étoiles et l'ajout de pratiques de définition de produit, couplée à un changement de culture, permet d'accéder à trois étoiles.

Nous y reviendrons dans le chapitre 22 *Transformer les organisations*.

Bien commencer

La question à se poser

Est-ce que nous avons bien conservé ce qui fait l'essence de Scrum ?

De mauvais signes

La contextualisation est réduite à s'aligner sur ce que propose l'outil informatique.

On a oublié les pratiques d'ingénierie.

Par quoi démarrer

Évaluer le contexte pendant le sprint zéro.

Une lecture pour tous

Le référentiel de l'Institut Agile [Institut, Référentiel].

À retenir

Scrum ne se vend pas en pack de 6. L'adaptation des pratiques Scrum au contexte et l'ajout de nouvelles pratiques sont les deux mamelles de son application sur un projet.

Pour cela, il convient d'évaluer régulièrement le contexte dans lequel se situe une équipe Scrum. La mise en évidence du contexte permet de savoir à quelles pratiques il faut prêter attention pour les adapter à la situation.

Références :

☞ Institut Agile, *Référentiel des concepts, pratiques et compétences agiles*, Web.

<http://institut-agile.fr/>

☞ Philippe Kruchten, *The Context of Software Development*, Web, 2009.

<http://philippe.kruchten.com/2009/07/22/the-context-of-software-developmen>

☞ James Shore et Diana Larsen, *Your Path through Agile Fluency*, Web, 2012.

<http://agilefluency.com>

Notes

[1] <http://www.noop.nl/2009/04/the-big-list-of-agile-practices.html>

[2] Visible sur le site Agile Alliance : <http://guide.agilealliance.org/subway.html>.

[3] Une présentation faite avec Philippe Kruchten lors de l'Agile Tour Toulouse 2008 : <http://www.aubryconseil.com/download/168>.

Découvrir le produit

Pour un nouveau produit, le défi est d'identifier les impacts souhaités sur les utilisateurs et les transformer en stories réalisables dans un sprint. Ce sont des activités de définition de produit. Nous sommes dans un domaine qui se situe en amont de Scrum : il faut donc ajouter des pratiques complémentaires.

Si Scrum aide à **bien** faire le produit, ces pratiques contribuent à faire le **bon** produit.

Les outils comme l'impact mapping et le story mapping, qui sera abordé dans le chapitre suivant, apportent une « big picture » du produit qui manque si on se cantonne à Scrum.

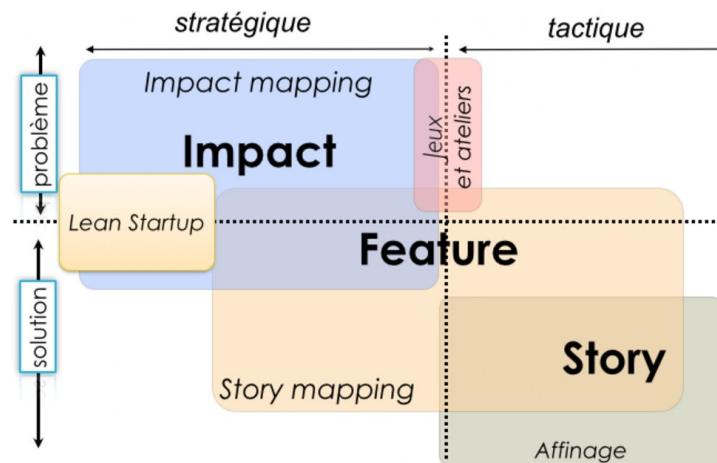


Fig. 14.1 Les éléments de la « big picture »

Dans la suite de ce chapitre, nous allons aborder les activités de découverte du produit permettant d'obtenir un premier backlog de features. Dans le chapitre suivant, nous verrons comment décomposer ces features en stories.

14.1. De l'idée aux features

Pour aller de l'idée initiale jusqu'aux features, nous allons nous poser les questions suivantes : « *Quelle est notre vision ? Qui sont les acteurs ? Quels seront les impacts ? Quelles features proposer pour obtenir les impacts ?* »

Attention, l'approche présentée ici pour affiner le produit peut paraître uniquement descendante (*top down*), des idées jusqu'aux stories. Il n'en est rien, on procède également de manière inductive.

14.1.1. L'outil fédérateur : impact mapping

L'impact mapping [Adzic, *Impact*] est une nouvelle technique, simple et efficace, de planification stratégique. Elle aide à aligner les travaux de l'équipe de développement avec les objectifs métier. C'est déjà une des raisons de la présence du Product Owner dans une équipe Scrum. L'impact mapping permet d'aller beaucoup plus loin en prolongeant l'alignement jusqu'à l'objectif recherché.

La carte des *impacts* se présente sous forme de carte heuristique (*mindmap*) qui contient, pour le produit envisagé :

- son objectif, au centre de la carte ;
- les acteurs au premier niveau ;
- les impacts attendus, au deuxième niveau ;
- enfin, les *features* qui sont la solution à ces impacts.

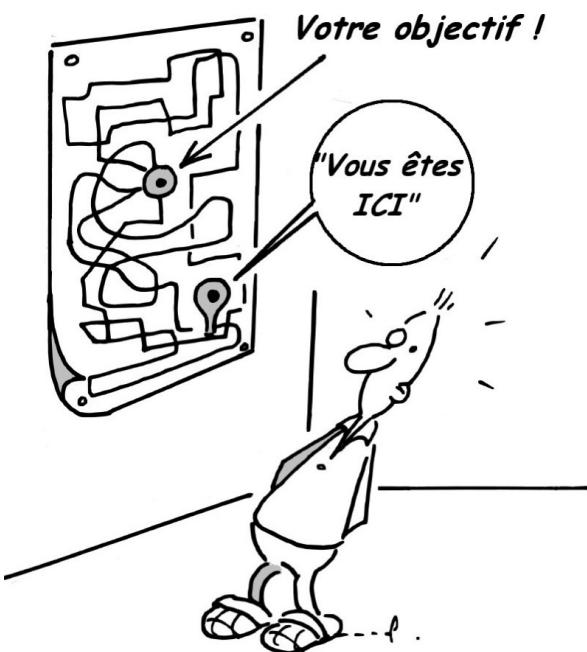


Fig. 14.2 Carte heuristique pour la découverte du produit

Cela donne une présentation du produit sous une forme synthétique et élégante (figure 14.3).

L'impact mapping peut s'appliquer pour découvrir la vision d'un produit ou bien d'une release.

14.1.2. Avec qui définir le produit ?

Le Product Owner joue un rôle important dans la définition du produit ; cependant il est souhaitable que son élaboration soit collective, au cours d'ateliers. Aux premiers vont participer des personnes métier, marketing, produit ; leur présence permettra d'obtenir des impacts selon différents points de vue.

L'équipe Scrum n'est pas toujours constituée à ce moment, mais il est recommandé que le Product Owner, le ScrumMaster et un développeur architecte soient déjà identifiés et participent aux ateliers.

14.1.3. Quand définir le produit

La définition du produit se fait au début de la vie du produit, ou lors de la transition à Scrum sur un produit existant. Elle porte alors sur la vision produit, puis on se concentre sur la première release.

Elle est ensuite effectuée au début de chaque nouvelle release, avec éventuellement un ajustement de la vision produit.

Elle a sa place dans le sprint zéro, sous formes de plusieurs ateliers. Le minimum est d'avoir au moins deux ateliers d'une demi-journée chacun. Elle précède le premier affinage du backlog, que nous avons présenté dans un chapitre précédent.

14.2. Définir la vision produit

La vision est l'expression, partagée, d'une volonté collective de développer un excellent produit ou service. Elle contribue à l'investissement de l'équipe dans le projet. La vision doit être ambitieuse pour entraîner les énergies et donner un élan, tout en restant réaliste.

La vision fait partie de ces notions dont on s'accorde pour dire qu'elles sont indispensables. Sur la forme qu'elle prend, les avis sont plus variés.

La vision faisait partie des artefacts majeurs de RUP (*Rational Unified Process*). À l'époque, c'était un document d'une dizaine de pages. Maintenant, la vision est généralement plus brève.

14.2.1. Formats de la vision

Une technique souvent utilisée, venant du marketing, est le test de l'ascenseur.

Tab. 14.1 Le pitch format ascenseur pour Peetic

Pour	Les propriétaires d'animaux domestiques
Qui	Aiment partager l'amour pour leur animal
Peetic	Est un site de rencontres en ligne
Qui permet	De faciliter les rencontres entre les animaux et entre leurs maîtres
À la différence de	Des rencontres au hasard
Notre produit	Apporte la possibilité de choisir les animaux et leurs maîtres.

Le produit est présenté en six points qui constituent un résumé permettant de comprendre rapidement la solution proposée. Ce nom vient de l'idée que l'argumentaire, destiné à convaincre son chef, doit être exprimé pendant le temps que prend un voyage en ascenseur avec lui.

La technique du « pitch » est utilisée dans les start up Week-Ends, comme je l'ai vu à Toulouse (<http://toulouse.startupweekend.org>) : la présentation de chaque projet est limitée à une minute et cela suffit au jury pour sélectionner les équipes autorisées à continuer pendant le week-end.

La vision peut aussi s'exprimer avec une simple phrase :

« Faire un site de rencontres pour animaux car les gens dépensent beaucoup d'argent pour leurs petites bêtes. Nous pourrons ainsi vendre de l'accompagnement, des produits et des publicités. »

Par rapport à notre tableau pour le test de l'ascenseur, on voit que le point de vue est différent : on est ici dans la peau du fondateur de Peetic.

L'impact mapping permet de considérer des points de vue d'acteurs différents et de les fédérer vers un même objectif.

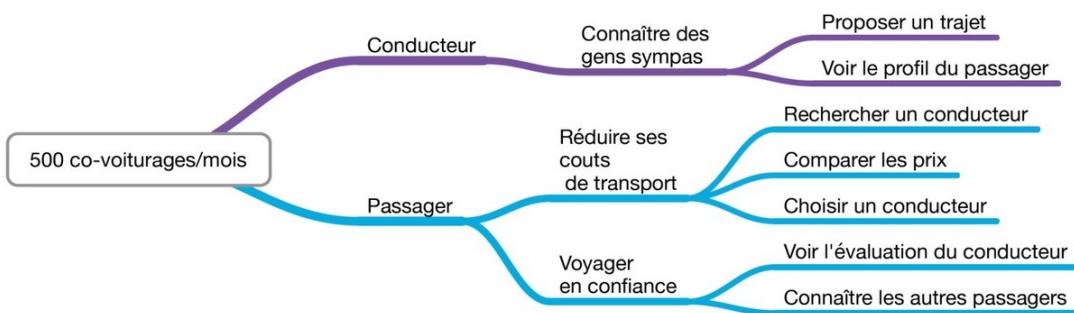


Fig. 14.3 Une carte qui donne la vision pour un site de co-voyage

14.2.2. Portée de la vision

La vision constitue le premier contact pour quelqu'un qui arrive sur le projet ou qui s'intéresse au produit. Elle s'énonce avec un langage simple : pour qu'elle soit comprise et partagée, il faut qu'elle soit bien résumée et bien écrite. Au début du développement d'un nouveau produit, on ne se pose pas de question : la vision décrit le produit. Le développement se faisant par des releases successives, doit-on remettre à jour la vision pour chaque release ? Doit-on élaborer une nouvelle vision ?

Après avoir essayé plusieurs façons de faire, je préconise maintenant de définir la vision une seule fois, au début de la vie du produit, éventuellement remise à jour à chaque release.

La vision présente le cap pour plusieurs releases. Les parties prenantes sont aussi identifiées avec le même horizon du produit. En revanche, nous allons associer la carte des impacts à chaque release.

14.3. Identifier les parties prenantes

Un produit est destiné à ses utilisateurs. Mais pas seulement, c'est pour cela qu'on élargit à toutes les personnes intéressées : les parties prenantes (Gojko Adzic utilise le terme « acteur »).



Fig. 14.4 Identifier les acteurs et les PP

14.3.1. Intérêt de connaître les parties prenantes

La connaissance des PP permet d'identifier leurs besoins afin de faire émerger les impacts attendus.

Il ne faut pas en rester à un seul type d'utilisateur, ce n'est pas assez précis, cela va confiner la réflexion à un seul point de vue.

Une façon d'aller plus loin est de réfléchir :

- d'abord aux utilisateurs primaires ;
- ensuite, aux utilisateurs secondaires, ou utilisateurs de deuxième rang, comme un administrateur ou un modérateur ;
- enfin à toutes les autres parties prenantes qui ont un intérêt dans le produit, sans en être des utilisateurs, comme le sponsor ou l'éditeur.

14.3.2. Identifier utilisateurs et parties prenantes

Pour identifier les utilisateurs, on se pose des questions comme :

- Qui utilisera le produit ?
- Qui assurera le support et la maintenance du système ?

La collecte est plus difficile pour des systèmes embarqués, puisqu'il n'y a pas d'utilisateurs en relation directe. Il est néanmoins utile d'identifier les utilisateurs plus éloignés et les interfaces.

Les parties prenantes qui ne sont pas des utilisateurs directs seront également identifiées. La réflexion sur les personnes qui ont de l'influence sur la diffusion du produit facilite leur découverte, suivie de leur priorisation [Gymkhana].

14.3.3. Personas

Pour affiner la notion d'utilisateur, la pratique des *personas* se révèle efficace.

Un persona est une représentation fictive, mais réaliste, de l'utilisateur final d'un produit ou d'un service.

Ces profils fictifs peuvent comporter des informations plus ou moins denses, à adapter selon le contexte.

Voici quelques exemples de la structure d'un persona type, dans le domaine du numérique :

- Son profil (âge, sexe, statut social, enfants, métier, type d'habitation, ville, etc.).
- Son utilisation des terminaux (mobile, tablette, desktop, laptop).
- Son utilisation des réseaux sociaux.
- Son comportement vis-à-vis du produit ou du service.
- Son attachement aux nouvelles technologies (qu'est-ce qu'il utilise dans son quotidien et pour quelle finalité ?).
- Ses attentes et frustrations potentielles vis-à-vis du produit.
- Sa perception du produit, résumée en trois points clés.
- Le potentiel du persona pour le produit, résumé en trois points clés.
- Son ambition, résumée en une phrase courte.

Ces informations peuvent être complétées par des statistiques, études, entretiens, et tout autre document améliorant la compréhension.



Fig. 14.5 Deux personas de Peetic

Les attentes et les besoins des personas servent de ligne directrice pour considérer toute l'évolution du produit ; ils ne seront probablement pas tous satisfaits en une seule release.

Pour construire les personas, on suit une approche qui favorise la collaboration et la priorisation : comme une phase d'idéation, impliquant idéalement l'équipe, chacun représente sa vision, puis on les co-construit en les confrontant

ensemble.

Si le produit évolue, il est pertinent d'enrichir nos personas au rythme de nos releases. C'est alors le moment d'ajouter de nouveaux profils auxquels on n'avait pas pensé dans un premier temps, mais qui se révèlent ceux des *early adopters* de notre produit, ou encore d'en supprimer, si finalement ils ne représentent pas des utilisateurs réceptifs.

Les personas, fictifs, servent à identifier les rôles d'utilisateurs et à déterminer lesquels sont les plus importants. Après les avoir définis, et avec la liste des features, on pourra mettre en place leur parcours à travers le produit. Cela fera l'objet du *story mapping*.

14.4. Définir le produit attendu en fin de release

Après avoir défini la vision produit et identifié les PP, on se focalise sur la prochaine release.

14.4.1. Trouver les impacts

On se pose la question « **Comment ?** ». Comment, dans notre vision, apporter un impact significatif sur la vie des PP ?

Un impact n'est pas une solution

Les pratiques classiques de définition de produit s'intéressent aux besoins des utilisateurs. Un document typique s'appelle « Expression des besoins utilisateurs ». La notion d'impact dépasse cette idée de besoin pour se focaliser sur le comportement de l'utilisateur.

Un impact est un changement dans le comportement d'une partie prenante.

À ce niveau, on ne s'aventure pas encore dans le domaine de la solution. On ne préjuge pas de ce qu'on va proposer. On peut d'ailleurs contribuer à un impact sans faire de développement.

Pour l'impact « faire des rencontres en promenant mon chien », on pourrait y arriver par un envoi de lettres incluant le bulletin d'inscription plutôt que par une solution informatique.

Identification des impacts

L'identification des impacts peut être facilitée par un atelier : « les impacts rétro-futuristes » [Gymkhana].

Je vais le décrire rapidement, mais il faut vraiment y jouer pour constater toute son efficacité.

On demande aux participants de se placer dans le futur en considérant que le produit est déployé depuis quelque temps et que c'est un succès. Ils jouent le rôle d'une des parties prenantes identifiées précédemment. Ils donnent leurs raisons pour dire que c'est un succès en revenant sur des faits concrets qui ont changé leur vie.

Chaque groupe prendra le point de vue d'un PP, ce qui permettra d'identifier les impacts, qui seront ajoutés sur la carte heuristique.

Visualiser les impacts sur la carte

La carte permet de comprendre comment un impact contribue à l'objectif.

Il est intéressant d'établir un ordre entre les impacts. Parfois c'est même au niveau des PP que la priorité est claire : on cherche à en satisfaire un en premier.

La notion d'impact contribue à s'intéresser au **changement** attendu dans la « vie » de l'acteur. Pour y aider, des éléments mesurables peuvent être associés aux impacts.

Exemple d'impact mesurable : faire trois rencontres de propriétaires de chien par mois. Une fois que le produit est déployé, on raisonnera plutôt en relatif : augmenter les ventes de bière pour chien de 20 %.

14.4.2. Définir l'objectif de la release

Le centre de la carte des impacts sert à répondre à cette question fondamentale : pourquoi ?

Dans notre approche, la portée est limitée à la release qui commence.

Identifier des objectifs

Les objectifs des projets sont dans la tête de ceux qui en sont à l'initiative. Quand ils sont écrits dans un document, ils sont nombreux et pour la plupart flous, c'est-à-dire qu'on pourra difficilement vérifier s'ils sont atteints.

Un objectif présente une hypothèse mesurable et vérifiable.

Choisir un objectif pour la release

L'impact mapping oblige à s'occuper d'un seul objectif par carte.

Il est préférable d'avoir identifié des impacts pour définir l'objectif. Mon expérience montre que c'est plus facile à ce moment-là.

L'objectif pour Peetic

Plusieurs objectifs ont émergé des discussions entre les parties prenantes sur les impacts :

- Vendre 12 niches par mois.
- Vendre 20 séances de coaching par des éleveurs.
- Avoir 1 000 inscriptions sur le site un mois après la mise en production.

C'est ce dernier objectif qui a été choisi pour la première release, qui se termine dans deux mois. Les trois premiers seront considérés à nouveau pour des releases ultérieures, uniquement si l'hypothèse est vérifiée.

14.4.3. En déduire les features

On se pose la question « **Quoi ?** ». Le niveau suivant sur la carte porte sur les features qui vont permettre d'actionner les impacts. Que pouvons-nous faire pour contribuer aux impacts attendus ?

Une **feature** est un service ou une fonction d'un produit dont l'énoncé est clair pour les parties prenantes ; une feature contribue à un impact et se décompose en stories.

Identification des features

La carte des impacts place les features en relation avec les impacts. Cela aide considérablement à les identifier, et évite de partir d'une liste de courses (les features) qui ne seraient pas alignées avec l'objectif.

La granularité d'une feature est arbitraire, peu importe à ce moment-là. Ce qui compte, c'est d'obtenir une première liste de cinq à quinze features.

Réflexion sur la stratégie de priorisation

Pour établir la priorité entre les features, les mêmes critères que pour les stories (valeur métier et valeur de connaissance, taille, dépendances) pourraient être utilisés.

Cependant, il est extrêmement difficile d'évaluer la valeur métier ou la valeur de connaissance d'une *feature* (lire à ce sujet le chapitre 18 *Améliorer la visibilité avec les indicateurs*).

Il existe bien des ateliers permettant d'estimer l'utilité des features de façon relative, sans unité. Mais cela reste subjectif. Il est tout aussi difficile d'estimer la taille des features au début de la vie du produit.

Avec l'impact mapping, la stratégie de priorisation repose sur des éléments moins sujets à interprétation.

En effet, les priorités définies sur les impacts induisent celles sur les features. Et comme ces priorités sont basées sur des objectifs mesurables, on évite la subjectivité de la notion de valeur, non vérifiable, pour les features. Avoir des objectifs vérifiables force à plus d'objectivité face à la situation.

Représenter les features sur la carte

En sélectionnant trois ou quatre impacts significatifs pour la release, on obtient cette première liste d'une dizaine de features. Elles sont ajoutées sur la carte des impacts, en relation avec l'impact auquel elles contribuent.

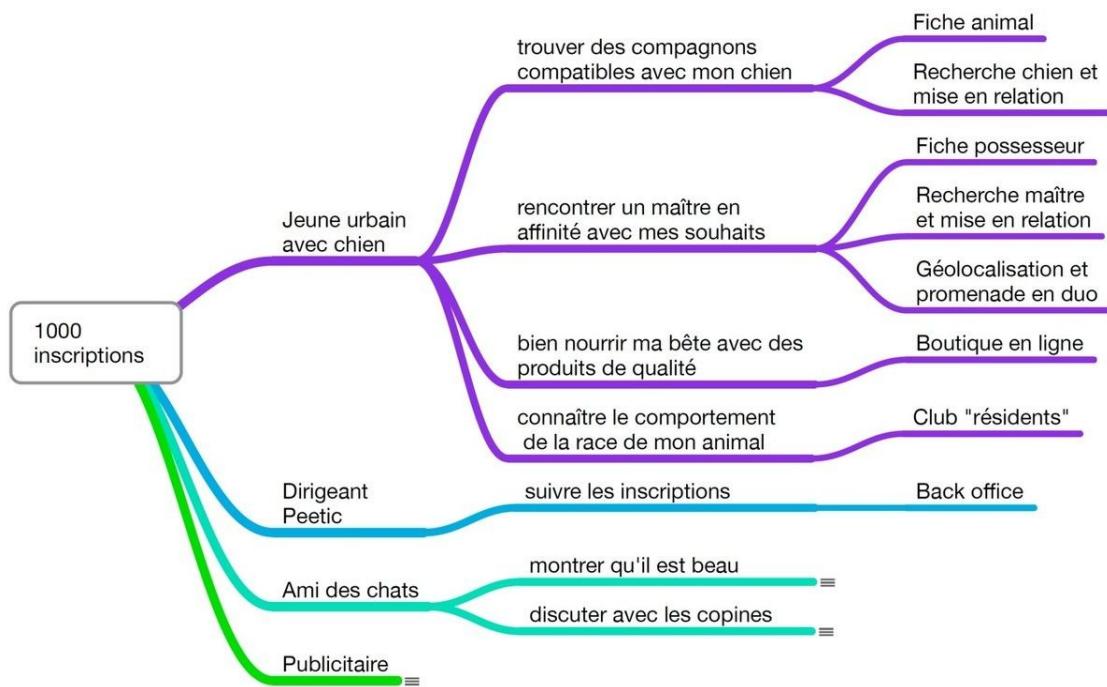


Fig. 14.6 La carte des impacts pour Peetic, avec les features

Le premier backlog est celui constitué par les features, ordonnées par priorité déduite des impacts. C'est le tableau de features dont nous avions parlé dans le chapitre 6 *Structurer le backlog*. Nous verrons comment arriver au backlog initial de stories dans le prochain chapitre.

14.5. La découverte du produit sur le terrain

14.5.1. Prioriser dès les impacts

Ce n'est pas une bonne idée de développer la carte des impacts niveau après niveau, par exemple pour avoir une approche exhaustive pour tous les PP d'abord, puis tous les impacts pour tous ces PP, etc.

Il est préférable de ne développer les impacts que pour les PP les plus importants pour l'objectif visé, puis de ne développer les features que pour les impacts les plus prioritaires.

Il est recommandé de procéder en deux étapes pour élaborer la carte [Adzic, *Impact*] : préparation d'abord, puis la cartographie. On consacre au moins une demi-journée à chacune (tableau 14.2).

Tab. 14.2 Les deux parties de l'impact mapping

1- Préparation	2 - Cartographie
Découvrez les vrais impacts.	Dessinez le squelette de la carte.
Identifiez les objectifs et essayez de n'en garder qu'un par release	Cherchez et ajoutez des alternatives.
Définissez les bonnes mesures.	Définissez les priorités.
Concentrez-vous sur la première release.	Gagnez ou apprenez (« <i>earn or learn</i> »).

14.5.2. Soigner l'ergonomie

L'ergonomie est une approche issue des sciences cognitives, visant à favoriser l'utilité et l'utilisabilité d'un produit ou d'un service auprès de ses utilisateurs, tout en leur donnant satisfaction.

L'utilité est le fait de répondre aux besoins et aux attentes des utilisateurs afin de les satisfaire, tandis que l'utilisabilité vise à satisfaire au mieux les besoins, avec une simplicité d'usage. On peut très bien avoir un site qui offre plein de fonctions, mais si on ne sait pas où trouver ce qu'on cherche, ou qu'on rencontre trop d'obstacles sur notre chemin, le produit ne sera pas ergonomique.

L'optimisation de l'ergonomie d'une application consiste donc à travailler sur l'adéquation des features aux attentes des utilisateurs (les impacts), ainsi que sur sa facilité d'utilisation. Les utilisateurs doivent réussir à obtenir l'impact attendu avec le moins de contraintes possibles, et leur satisfaction déterminera une ergonomie réussie.

L'ergonomie est basée sur des principes, en voici quelques-uns :

- La théorie de la Gestalt indique que, devant plusieurs formes, le cerveau humain est capable de distinguer ou d'assembler différents éléments comme un groupe et non comme des entités individuelles ou inversement.
- La loi de Fitts énonce qu'une cible est d'autant plus rapide à atteindre qu'elle est proche et grande. Plus la cible est éloignée et petite, plus ce sera long de l'atteindre, et le taux d'erreur sera également plus élevé. Une recommandation consiste à augmenter la taille des éléments cliquables, par exemple.
- Le « principe du 7, plus ou moins 2 », ou loi de Miller, stipule que l'être humain peut retenir environ sept éléments à la fois dans la mémoire à court terme.

Il ne suffit pas de les appliquer pour qu'une ergonomie soit réussie, dans la mesure où chaque produit est unique. Il faut penser l'ergonomie à travers les impacts sur les personnes.

14.5.3. Penser à l'expérience utilisateur

Le *design* d'expérience utilisateur (*UX design*) est une nouvelle manière de penser les dispositifs numériques. C'est une démarche pragmatique, pluridisciplinaire, orientée vers la résolution des problèmes et vers l'innovation. Cette approche puise ses racines dans le *design* centré sur l'utilisateur et le *design thinking*, en continuant d'évoluer et de s'enrichir de méthodes provenant d'autres disciplines (design de service, etc.).

Le design d'expérience utilisateur est un processus de recherche, d'imagination, de conception, de création, de test et d'optimisation. Il implique tous les métiers concernés par la réalisation du produit.

L'UX est centrée sur le produit, l'utilisateur étant au cœur de la stratégie. On privilégie l'efficience et la satisfaction de l'utilisateur. C'est donc cohérent avec une approche agile comme Scrum, qui permettra un feedback rapide des utilisateurs. À la fin d'un sprint, le

résultat permettra d'apprendre, apportant de la valeur de connaissance. Les définitions de prêt et de fini aideront à préciser les attendus d'une story d'expérimentation d'UX.

14.5.4. S'inspirer du Lean Startup

Le Lean Startup est une démarche de développement produit, au sens « marketing » et non « logiciel », basée sur une approche pragmatique d'apprentissage par les retours des utilisateurs [Deverge, *Lean Startup*].

Le principe peut être résumé de la manière suivante : on va essayer de dépenser le moins d'énergie possible pour valider ou invalider un certain nombre d'hypothèses (idée, solution, marché cible, modèle économique, etc.) auprès d'une population d'utilisateurs intéressés, en se basant sur des mesures concrètes [Maurya, *Running Lean*].

Le Lean Startup propose plusieurs outils :

- Le *Lean Canvas* permet de formuler les différentes hypothèses, c'est un modèle de tableau à remplir afin d'obtenir une vision synthétique du produit et du marché auquel il s'adresse.
- Le MVP (*Minimum Viable Product*) va permettre de tester les hypothèses, c'est un prototype du produit, pas un vrai produit, mais une expérimentation fonctionnelle qui nécessite très peu d'investissement et permet d'avoir un retour rapide de la part des (futurs) utilisateurs.

Le Lean Startup se situe dans l'esprit des méthodes agiles, avec lesquelles il partage l'importance donnée au feedback rapide.

Un « canevas » peut être proposé, comme alternative à l'impact mapping, pour définir le produit avec des parties prenantes.

Bien commencer

La question à se poser Est-ce que nous faisons le bon produit ?

De mauvais signes
Des impacts qui sont en fait des features.
Des solutions détaillées trop tôt.

Par quoi démarrer Un atelier ludique avec toutes les parties prenantes pour lancer un impact mapping.

Une lecture pour tous *Spécifiez Agile* [Cros, *Spécifiez*], une autre façon d'aborder la définition de produit, qui puise ses origines dans Extreme Programming.

À retenir

Scrum permet de *bien développer le produit* demandé par le Product Owner. Pour *développer le bon produit*, celui qui répond à l'objectif défini par les parties prenantes, l'impact mapping apporte une vision stratégique et permet d'aligner le développement sur des impacts métier.

Pour un nouveau produit, l'impact mapping, complété avec quelques ateliers, contribue à la constitution d'un premier backlog, constitué de features. La démarche consiste à s'intéresser aux utilisateurs et à leurs personas, à identifier les impacts attendus sur leur comportement et à en déduire les features qui y répondent. C'est une approche participative basée sur du feedback rapide.

Références :

☞ Gojko Adzic, *Impact Mapping* 2012.

<http://impactmapping.org>

☞ Thierry Cros, *Spécifiez Agile*, 2013.

<https://leanpub.com/agile-expression-de-besoins>

☞ Nicolas Deverge, *Le Lean Startup*, Web, 2015.

<http://www.aubryconseil.com/post/Le-Lean-Startup>

☞ Gymkhana, 2015.

<http://www.gymkhana.fr/>

☞ Ash Maurya, *Running Lean*, VF, 2012.

<http://www.diateino.com/fr/95-la-methode-running-lean-9782354561239.html>

Raconter la story

La story a une place centrale. La notion de story provient d'*Extreme Programming* : une *user story* y est présentée comme un rappel pour une conversation, lancé dans le but de faciliter la planification.

Nous avons généralisé cette notion en story, élément du backlog qui apporte de la valeur à quelqu'un et qui sera développé en un seul sprint.



Fig. 15.1 L'art du storytelling.

Nous avons vu qu'elle était affinée pour être prête, puis qu'elle était réalisée pour être finie dans un sprint.

Nous allons apporter des compléments à des activités délicates et importantes qui portent sur la story pour montrer comment :

1. identifier les stories à partir des features,
2. décomposer des stories épiques,
3. ajouter une condition d'acceptation,
4. accepter la story.

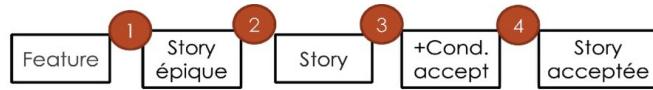


Fig. 15.2 Les activités pour raconter la story

La première activité complète le chapitre 14 *Découvrir le produit* pour aboutir au premier backlog de stories. La dernière permet de finir la story pendant l'exécution du sprint. Les

deux autres font partie de l'affinage.

15.1. Identifier des stories avec le story mapping

15.1.1. Introduction au story mapping

Dans le chapitre précédent, nous avons montré comment arriver à une liste de features. Nous allons maintenant passer à la suite de la découverte avec les stories.

Pour cela nous allons nous appuyer sur l'atelier story mapping, une pratique popularisée par Jeff Patton [Patton, Story mapping].

La pratique du *story mapping* est devenue incontournable. Dans son livre, Jeff Patton met en évidence qu'une liste de stories dans un backlog, c'est « plat ». La tendance est à avoir des stories très petites, ce qui présente des avantages en termes de gestion et de suivi. Mais cela a l'inconvénient de rendre l'ensemble plus difficile à comprendre. Une story est à replacer dans un contexte plus large pour que l'on comprenne son usage.

En fait, une story ne suffit pas toujours pour raconter une histoire qui parle aux parties prenantes. Une story map permet de visualiser des scénarios d'usage et de préparer l'enchaînement des stories.

Les deux dimensions considérées pour élaborer la carte sont la séquence en horizontal et la nécessité en vertical.

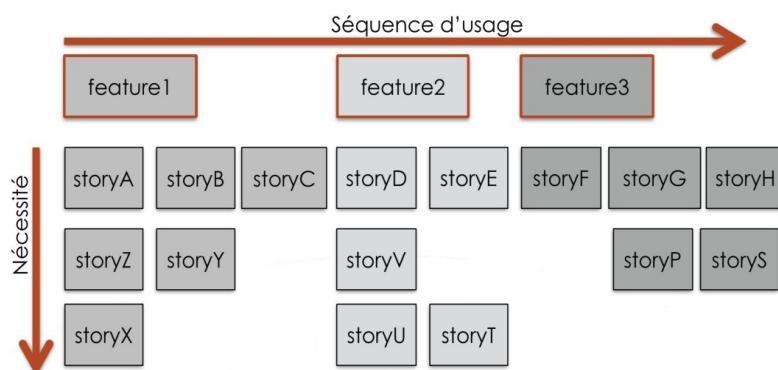


Fig. 15.3 Vue partielle d'une story map

Nous allons adapter le story mapping à notre usage, en partant de la liste des features obtenue avec la carte des impacts, présentée dans le chapitre précédent.

Nous y avons dressé la liste des features prioritaires : entre 5 et 15.

Il est possible d'utiliser le story mapping directement, sans passer par la case impact mapping. On peut aussi l'utiliser différemment de ce qui est présenté ici, pour préparer une « roadmap » avec plusieurs releases^[1].

L'atelier se déroule pendant le sprint zéro lors de l'affinage initial. Il prend de quelques heures à une journée, selon le contexte.

La carte est élaborée sur un mur, une table ou carrément sur le sol, avec des Post-it®. L'intérêt de la représentation spatiale est qu'elle permet une élaboration collective. Toute

l'équipe y participe.

15.1.2. Maximiser l'impact, minimiser l'effort

L'atelier permet de préparer la release qui commence. Son contenu, sans être détaillé, sera suffisamment éclairci pour converger sur l'objectif de la release. À la priorisation par les impacts, le story mapping ajoute la prise en compte des dépendances et des risques.

L'objectif principal est d'éviter de faire des choses qui apportent peu de valeur pour la prochaine release. Le story mapping permet de tailler le produit sur deux axes :

- en coupant les features non indispensables,
- parmi celles qui restent, en enlevant les stories non nécessaires.

15.1.3. Dérouler un story mapping

Obtenir la colonne vertébrale

La première étape du story mapping consiste à créer la colonne vertébrale de l'application sur la première ligne. Les features sont organisées de façon chronologique, en vue de définir un enchaînement (une chaîne de valeur) logique selon l'usage du produit par ses acteurs. C'est l'axe de la séquence, qui rappellera les *use cases* à certains.

On ne passera pas trop de temps à discuter de l'ordre dans la séquence, l'objectif est d'avoir une première ligne de features en s'arrêtant à celles qui sont obligatoires dans la première release.

Décomposer les features en stories

Ensuite, on part de la feature la plus prioritaire pour la décomposer en stories dont chacune est enregistrée sur un Post-it®. Pour une feature, il peut y avoir plusieurs stories, jusqu'à une dizaine. Elles sont rangées sur plusieurs lignes en dessous : les lignes du haut pour les stories les plus nécessaires. Cette question de nécessité est posée au PO dans ces termes : « *Est-ce que tu en as vraiment besoin dans une première version ?* ». Si c'est oui, elle reste sur la ligne du haut, sinon elle est placée plus bas.

On continue avec les features suivantes. Si l'équipe est nombreuse, j'encourage à un essaimage pour paralléliser le découpage en stories : par exemple, si le squelette comporte huit features et l'équipe huit personnes, des binômes se constituent et chacun prend deux features. Une restitution collective des travaux est indispensable dans ce cas.

Le but est d'obtenir sur les premières lignes de stories, en dessous des features, l'ensemble minimal permettant aux acteurs d'utiliser le produit.

Ajouter les risques

On demande aux développeurs d'indiquer les stories qui leur paraissent présenter des risques technologiques.

Ils ajoutent un Post-it® de couleur différente sur la carte, près des stories qui comportent le risque identifié.

15.1.4. Obtenir la feature minimale

Après avoir identifié les stories correspondant à une feature, leur affinage amène à les décomposer, mais aussi à tailler dans la feature initiale.

Exemple Peetic : il s'agit de proposer du coaching en ligne par un éleveur.

L'idée est d'offrir un premier conseil et de faire payer les suivants.

L'équipe a identifié les stories suivantes : le client demande un conseil gratuit, l'éleveur répond le client achète dix conseils, il gère ses demandes...

Le Product Owner considère qu'il faut d'abord lancer le service gratuit pour attirer du monde. Dans la première release, il se contente des deux stories y correspondant : demander un conseil et y répondre. Cela constitue pour lui la feature minimale utilisable. Les autres stories peuvent attendre la release suivante dans le bac à glace.

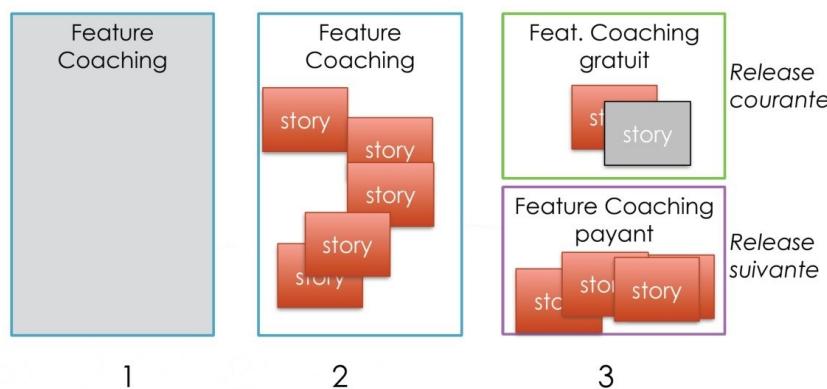


Fig. 15.4 Décomposition d'une feature (1) en stories (2) et regroupement en une feature minimale (3)

On obtient donc une *feature Coaching gratuit* qui sera finie dans la release en cours, et l'autre *feature Coaching payant* qui fera l'objet de la release suivante.

Il convient de s'assurer que l'impact attendu puisse quand même être obtenu avec cette réduction de voilure.

Cette décomposition peut être faite lors de l'élaboration de la story map, ou plus tard, proposée par le PO et discutée lors d'une séance d'affinage.

15.1.5. Alimenter le backlog

Cet atelier visuel et collaboratif permet d'identifier quelques dizaines de stories en une session. À la fin de la réunion, les stories pourront être introduites dans le backlog.

L'ordre défini avec les features et stories est conservé pour l'introduction des stories qui iront directement dans la partie bac d'affinage. Les features retourneront dans leur tableau. On aura fait le tri entre celles indispensables pour la release et les autres.

L'intérêt de l'approche est d'obtenir rapidement un backlog initial qui prend en compte les dépendances et les risques, en plus de l'impact métier.

15.2. Décomposer

Cette activité a lieu lors des séances d'affinage, dont nous avons déjà parlé dans un chapitre précédent.

15.2.1. Intérêt des petites stories

Pour être suffisamment petites et finies en un sprint, les stories doivent être décomposées. La story prête, dans le bac de départ, est « élémentaire ». Dans le bac d'affinage, il peut donc y avoir des stories épiques plus grosses qui ont vocation à être décomposées.

Arriver à de petites stories demande des efforts, mais procure des bénéfices :

- Les conditions d'acceptation sont plus faciles à identifier à travers la communication entre le PO et les développeurs.
- Cela apporte plus de flexibilité dans la planification car les stories décomposées peuvent être planifiées dans des sprints différents.
- Les stories étant moins variables en taille, on peut simplement utiliser le nombre de stories pour mesurer la vélocité.
- Elles sont finies plus vite et permettent un feedback rapide.

Petite, mais avec du sens : pour qu'une user story soit justifiée, il convient de s'assurer qu'elle correspond à une tranche verticale, en référence aux architectures de logiciel en couches. Une tranche verticale signifie que la story part de l'utilisateur, traverse toutes les couches (de l'IHM jusqu'à la base de données) pour remonter jusqu'à lui.

15.2.2. Techniques de décomposition

Il existe quelques techniques usuelles pour décomposer une story trop grosse. Cependant leur usage est contextuel, il s'agit de s'en servir à bon escient.

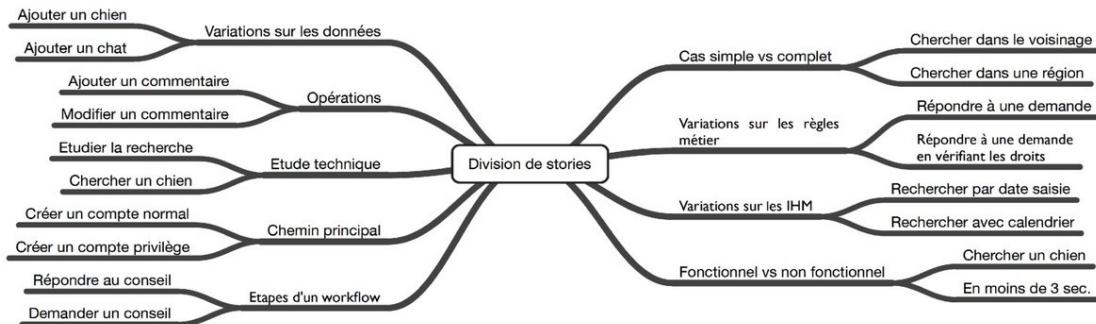


Fig. 15.5 Techniques de décomposition d'une story

Nous n'en présentons que quelques-unes. Il en existe bien d'autres [Adzic, User stories].

Variation sur les données

Exemple avec la story « Ajouter un animal à mon profil »

S'il existe plusieurs types d'animaux et que la façon de les créer n'est pas la même pour tous, on crée autant de stories qu'il y a de types d'animaux : ajouter

un chien, ajouter un chat (on ne le promène pas), ajouter un caméléon (sans photo), etc.

Une remarque fréquente est que les développeurs risquent de refaire leur travail si on ne pense pas dès le début à mettre en place une conception générique. En fait, rien n'empêche d'y réfléchir dès la première story. Mais pas besoin d'anticiper exagérément, l'agilité favorise la conception émergente.

Décomposition selon les règles de gestion

Dans le cas où des règles limitent l'accès à une fonction, on peut définir une story qui ignore ces règles, puis d'autres stories pour les prendre en compte. La première story permet d'apprendre, elle apporte de la valeur de connaissance.

Exemple : on peut proposer de rechercher les ancêtres d'un chien à pedigree, mais le réservier aux maîtres qui ont acheté pour plus de 100 € de croquettes et qui sont membres du club Peetic depuis plus d'un an.

Première story : Rechercher les ancêtres de mon chien.

Deuxième story : Restreindre la recherche à ceux autorisés.

Spike

Dans le cas où les développeurs font remonter qu'il y a plusieurs solutions techniques pour réaliser la story, on la décompose en deux parties : une première pour étudier les solutions et en choisir une, la seconde pour réaliser avec la solution choisie.

L'étude, appelée un « spike », permet de réduire les risques sur sa réalisation.

Décomposition selon fonctionnel / non fonctionnel

Dans le cas où cela donne une story trop grosse pour être finie en un sprint et s'il n'y a pas de risque majeur, on peut décider de faire dans un premier sprint la story sans se soucier de performance, puis de traiter dans un sprint ultérieur, avec une autre story, la conformité à l'exigence non fonctionnelle. C'est une des possibilités de décomposition de story, à utiliser seulement si l'équipe estime que cela a du sens.

Exemple : on peut avoir une story de recherche qui a pour but premier de fournir les résultats, puis une autre qui aura pour objectif d'optimiser la recherche pour afficher les résultats en moins de deux secondes.

15.2.3. Décrire une story avec son plan type

La manière la plus utilisée pour décrire une story est :

En tant que <acteur>, je veux <un but> afin de <une justification>

Pour le but, il faut penser à l'intention fonctionnelle. La justification porte sur le bénéfice apporté à une partie prenante.

En tant que membre du club Peetic, **je veux** demander un conseil à un éleveur **afin de** savoir quoi faire avec mon toutou qui aboie tout le temps.

En tant qu'organisateur d'événement canin, **je veux** connaître le nombre de chiens inscrits à la confirmation^[2] de bassets hound **afin de** choisir la salle adéquate en longueur.

L'utilisation de ce plan type n'est pas obligatoire. Je conseille d'essayer quand on débute.

Pour savoir, *in fine*, si une story doit être décomposée, on rappellera des idées déjà évoquées. La plus simple est l'estimation relative, qui permet de se rendre compte si une story a une taille trop importante. La plus avancée est l'association d'une story à son storyotype, qui montre si les vérifications induites (par exemple les « 6D ») sont pertinentes sur la story en question.

15.3. Ajouter une condition d'acceptation

Lors de l'affinage, pour que la story devienne prête, le PO et l'équipe ajoutent une **condition d'acceptation**.

Une story devrait posséder une condition d'acceptation, éventuellement plusieurs, mais pas trop cependant.

Un nombre trop important de conditions est le signe d'une trop grande complexité de la story, qu'il conviendrait de décomposer.

15.3.1. Acceptation et test

Selon la définition de prêt pour l'équipe, cela se fera pendant l'affinage ou pendant le sprint dans lequel la story est réalisée. On peut souhaiter avoir les tests écrits pour que la story soit prête, mais en général on se contentera d'avoir une condition d'acceptation.

Scrum met l'accent sur l'équipe sans spécialiser les rôles. Il n'y a pas de rôle explicite de testeur, mais cela ne veut pas dire que l'équipe ne teste pas ! Certains restent sur l'idée que c'est le client qui teste, ce qui peut conduire les développeurs à déléguer au PO tout l'effort de test.

Ce n'est pas une bonne idée : pour des raisons de quantité de travail et de compétences, le PO n'est généralement pas en situation de s'occuper seul de l'acceptation ; en outre, cela doit être un travail collectif, essentiel pour la communication.

Finalement, peu importe qui rédige les tests, ce qui compte c'est que tout le monde se comprenne et que cela soit fait au bon moment. Cela devient une responsabilité collective.

15.3.2. Identifier une condition d'acceptation

La première réflexion à faire est « *How to demo ?* », ce qu'on va montrer lors de la démonstration de cette story. Cela permet d'imaginer un scénario et d'aboutir à une condition d'acceptation.

Exemple avec la story « Inscription à une confirmation ». On peut identifier une condition de succès :

- **Inscription acceptée** – C'est le cas de succès, l'inscription d'un chien à une confirmation est validée.

Une autre condition est un cas d'échec.

- **Inscription refusée** – L'inscription est refusée, la salle ayant atteint sa limite.

Cette deuxième condition peut être associée à la même story ou bien faire l'objet d'une nouvelle story, c'est tout l'art de la décomposition.

15.3.3. Formuler la condition avec le BDD

Une condition d'acceptation se réfère à l'exécution de la story. Le comportement de la story quand elle est exécutée dépend de l'état de départ et des paramètres de l'exécution. La pratique BDD (*Behaviour Driven Development*) [North, BDD] permet de décrire ce comportement, avec une machine à états.

Chaque test est formalisé avec trois rubriques :

- l'état avant l'exécution du test (on parle aussi de précondition ou de contexte du test) ;
- l'événement qui déclenche l'exécution de la story ;
- l'état après l'exécution (on parle aussi de postcondition ou de résultat attendu).

Le formalisme textuel du BDD est le suivant :

- **Étant donné** le contexte **et** la suite du contexte.
- **Quand** l'événement.
- **Alors** résultat **et** autre résultat.

En anglais, cela donne « *Given When Then* ».

Cette façon de faire est particulièrement adaptée à des applications interactives. Elle pousse à avoir des tests courts, puisqu'on y décrit la réponse à un seul événement, celui qui déclenche la story.

Exemple avec la story « Inscription à une confirmation »

En tant que maître de chien de race, j'inscris mon chien à une confirmation afin de soigner son pedigree.

On peut utiliser le BDD pour la condition d'acceptation :

Étant donné un maître de chien de race **et** un événement de confirmation prévu pour cette race.

Quand le maître inscrit un chien d'un âge autorisé à une confirmation.

Alors il est informé de son inscription **et** le nombre des inscrits est incrémenté.

Un conseil pour s'y retrouver avec les clauses est de mettre le texte de « étant donné » au passé, celui de « quand » au présent et celui de « alors » au futur.

En donnant un exemple concret avec des données, on passe au test d'acceptation :

Étant donné Corinne propriétaire de Gary **et** la confirmation pour la race Basset Hound annoncée pour le 24 avril avec le nombre d'inscrits à 34.

Quand Corinne inscrit son basset Gary de 2 ans à la confirmation du 24 avril.

Alors l'inscription de Gary sera acceptée **et** le message « *Vous êtes bien inscrit à la confirmation du 24 avril* » sera envoyé à Corinne **et** le nombre des inscrits passera à 35.

La différence entre une condition et un test d'acceptation, c'est que le test porte des valeurs, qui constituent un exemple. C'est pour cela qu'on parle de **spécification par l'exemple** avec l'ensemble story plus tests d'acceptation.

Des outils orientés BDD se diffusent, comme Cucumber, pour faciliter la transformation de cette expression en tests automatisés.

Mais le BDD est avant tout une démarche visant à utiliser le langage du métier pour favoriser la communication entre le PO et les développeurs.

15.4. Accepter la story

Lors de la réunion de planification du sprint, PO et équipe ratifient l'accord sur ces conditions. Pendant le sprint, la story est prise en main par un essaim de développeurs. Les conditions d'acceptation peuvent être complétées au cours de **conversations**.

Une condition est **vérifiée** dès que possible. Lorsque tout est vérifié, la story est acceptée et **déclarée finie**. Pour éviter les régressions pendant et après le sprint, les tests doivent être rejoués, d'où l'importance de l'automatisation.

15.4.1. Continuer la conversation

L'ensemble des stories avec leurs conditions et tests d'acceptation remplace une spécification fonctionnelle détaillée, avec un bénéfice essentiel : la communication se fait directement.

Cela ne signifie pas qu'on ne garde pas de trace. En fait, ce sont les tests plutôt que les stories qu'il faut conserver comme référentiel. Le référentiel des tests est complété progressivement et toujours à jour.

Pendant le sprint, il est fréquent que des tests soient modifiés et complétés, voire que de nouveaux soient ajoutés, notamment au cours des différentes conversations avec le Product Owner. Celui-ci rejoint fréquemment un essaim pour aider à finir une story : c'est un butineur occasionnel.

Pour mettre en évidence l'acceptation de chaque story, on peut identifier une tâche et la faire figurer dans le tableau des tâches d'un sprint. Pour aller plus loin, on peut mettre les tests dans le tableau à la place des tâches : les Post-it® associés à une story feront alors référence aux tests ou plus largement aux conditions d'acceptation.

15.4.2. Vérifier dès que c'est possible

Le développement de la story est mené rapidement pendant le sprint ; il dure de un à trois jours à plusieurs personnes faisant partie de l'essaim.

Pour vérifier, il faut exécuter ses tests sur la dernière version du logiciel. Si des tests ne passent pas, la correction est faite vite (après avoir ajouté un Post-it® dans le tableau), l'objectif minimal étant que tous passent avant la fin du sprint. Un objectif plus ambitieux consiste à les repasser plusieurs fois pendant le sprint, à chaque fois qu'une story est finie, voire à chaque modification du code.

À chaque nouvelle version, pour éviter les régressions, il conviendrait donc de repasser tous les tests. C'est la raison pour laquelle il est nécessaire de s'intéresser à leur automatisation.

15.4.3. Accepter la condition et déclarer la story finie

Au vu des résultats de l'exécution des tests et en tenant compte des vérifications de la définition de fini, le Product Owner déclare la story finie ou, le cas échéant, signale ce qui ne va pas à l'équipe.

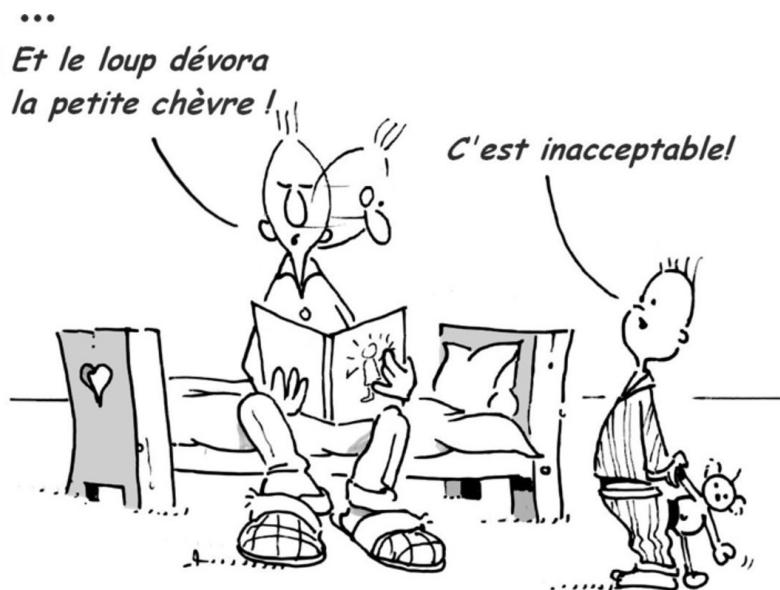


Fig. 15.6 La fin de la story n'est pas toujours acceptée.

Cela peut être délégué, en cas d'essaimage, ou si le PO n'est pas disponible pour le faire rapidement.

Risque : ne pas finir la story dans le sprint

Un des constats fait en suivant des équipes Scrum qui débutent est que de nombreuses stories ne sont pas finies en un sprint. Quelques-unes durent même plusieurs sprints !

Ce problème est souvent dû à l'accostage développeurs-testeurs. Si un testeur reçoit le logiciel à tester en toute fin de sprint, au mieux il découvre des erreurs qui ne pourront pas être corrigées avant la fin du sprint, au pire il diffère ses tests au sprint suivant. La pratique du BDD et l'essaimage permettent d'éviter ce problème.

Ne pas développer, tester et corriger une story dans le même sprint est un dysfonctionnement sérieux auquel il faut s'attaquer. Pourquoi est-ce un problème ?

- Cela diminue la productivité des développeurs, qui doivent se replonger dans le code implémentant une story qu'ils ont développée dans une itération précédente.
- Ce n'est pas satisfaisant pour l'équipe, y compris le PO. Elle s'est engagée au début de l'itération à finir une story, et le résultat montre que ce n'est pas fini.
- Cela rend la planification plus difficile. Une story non finie est comptée zéro pour la vélocité, alors que du travail a été effectué dessus. Cette dé-corrélation entre résultat et travail tend à produire un burndown chart de release en dents de scie, ce qui peut être perturbant.

Bien commencer

Est-ce que nous connaissons la colonne vertébrale de notre application ?

Les questions à se poser

Est-ce que nous arrivons à obtenir des petites stories, en gros 2-3 jours en moyenne ?

Est-ce que nous avons vraiment, dans l'équipe, des conversations à propos d'une story ?

Les stories des features prévues dans des releases ultérieures sont déjà identifiées. C'est une décomposition trop précoce.

Les stories sont trop grosses.

Les stories sont décomposées trop tôt.

La décomposition va trop loin, ce sont des tâches.

De mauvais signes

Une colonne « à tester » dans le tableau. C'est le signe que la vérification et le test ne se font pas dans la foulée du développement.

La condition d'acceptation trop longue, porte sur plusieurs stories.

Il y a des « et » dans une clause Quand du BDD.

	Les conditions d'acceptation sont difficiles à maintenir à cause de leur faible niveau d'abstraction.
Par quoi démarrer	Faire du story mapping. Commencer par réfléchir au « How to demo » pour une story.
Deux lectures pour tous	<i>Le Story Mapping</i> [Patton, <i>Story Mapping</i>] <i>Fifty quick ideas to improve your tests</i> [Adzic, Tests].

À retenir

La story est l'élément clé du développement par sprint. Lors de l'affinage initial pendant le sprint zéro, les stories sont identifiées. Une bonne façon de faire est de partir des features avec un atelier story mapping.

Au cours des séances d'affinage, une story est décomposée et approfondie, en général avec une condition d'acceptation.

Au cours du sprint où elle est développée, une story est acceptée rapidement grâce à la pratique de l'essaimage.

Références :

- ☞ Gojko Adzic & David Evans, *Fifty quick ideas to improve your User Stories* 2014, <http://www.50quickideas.com/>.
- ☞ Gojko Adzic & David Evans, *Fifty quick ideas to improve your Tests* 2015, <http://www.50quickideas.com/>.
- ☞ Dan North, Introduction au BDD, Web, 2006, VF.
<http://philippe.poumaroux.free.fr/index.php?post/2012/02/06/Introduction-au-Behaviour-Driven-Developement>
- ☞ Jeff Patton, *Le Story Mapping*, 2015, Dunod.

Notes

- [1] Exemple <http://www.areyouagile.com/images/2015/05/user-story-mapping.jpg>
- [2] La confirmation du chien permet à celui-ci d'être inscrit au LOF (Livre des origines français).

Planifier la release

Ce chapitre *Planifier la release* était placé bien plus tôt dans les précédentes éditions de mon livre. Il incorporait des activités d'affinage du backlog que j'ai regroupées dans un chapitre dédié. Je l'ai déplacé après les pratiques au cœur de Scrum. Car la planification de release, aussi importante soit-elle, n'est pas obligatoire. Le petit guide Scrum 2013 rappelle son caractère optionnel. Cependant, dans la très grande majorité des cas que j'ai rencontrés, la planification de release était une pratique nécessaire.

Ceux qui ne connaissent pas bien les méthodes agiles pensent parfois, à tort, qu'elles ne permettent pas de planifier, parce que « *ça change tout le temps* ». Ils ont bien compris que le client pouvait faire des changements et en déduisent trop vite : « *À quoi sert de faire des plans s'ils sont remis en question sans arrêt ?* ».

D'autres qui appliquent Scrum savent planifier le sprint, mais ne savent pas encore comment planifier la release. Pourtant, agile ou pas, il est souvent utile d'avoir des prévisions sur le moyen terme.

Dans le cadre de Peetic, des questions ne manquent pas de se poser à propos du futur, comme par exemple :

- *Pourrions-nous utiliser la gestion des inscriptions en ligne pour le salon du chien en mars ?*
- *Dans combien de temps pourrions-nous annoncer l'application sur tablette ?*
- *Quel est le budget nécessaire pour développer la release 2 ?*
- *Quand aurons-nous besoin du composant de paiement en ligne, qui est développé à l'extérieur ?*

Ce chapitre montre comment la planification de release permet d'obtenir des réponses à ces questions.

16.1. Pourquoi planifier plus loin que le sprint ?

16.1.1. Prévoir pour préparer le déploiement du produit

La release est une série de sprints. Nous avons vu à quoi servait la planification de sprint et comment elle se déroulait. Planifier la release, c'est essayer de prévoir le résultat à la fin de cette série. L'horizon est plus lointain. À quoi cela sert ? D'abord, si le déploiement coïncide avec la fin de release, à informer les parties prenantes, qui attendent le produit ou y contribuent.

16.1.2. Prévoir pour se synchroniser

Rappelons que la fin de la release ne coïncide pas toujours avec le déploiement d'une version. Dans « planification de release », le terme release est utilisé comme période de temps.

Le déploiement auprès des utilisateurs finaux ne se fait pas toujours en fin de la release :

- d'un côté, il y a des systèmes industriels pour lesquels le déploiement ne se fera peut-être qu'après plusieurs releases ;
- d'un autre côté, pour des applications web, le déploiement peut être fait plus fréquemment, à chaque fin de sprint, voire plus souvent en cas de « déploiement continu ».

Ceci étant dit, quel que soit le type de déploiement auprès des utilisateurs finaux, la planification de release reste une pratique permettant de prévoir des points de rencontre avec des parties prenantes, notamment pour obtenir leur feedback, ou pour se synchroniser avec d'autres équipes.

La planification de release ne concerne pas que l'échéance finale. Prévoir les prochains sprints est utile pour définir les synchronisations nécessaires avec d'autres équipes ou d'autres personnes.

16.1.3. Prévoir pour décider

La meilleure façon de procéder pour la release est de définir une date de fin et de s'y tenir, en reprenant l'idée de la timebox, comme pour le sprint.

La **release à date fixée** définit le budget à l'avance et un horizon pas trop lointain, ce qui impose au PO de prendre des décisions :

- sur les priorités des éléments du bac d'affinage ;
- sur la purge de stories ne présentant finalement que peu d'intérêt.

Le plan de release, en présentant la situation lors de la revue de sprint, permet de prendre des décisions avec les parties prenantes sur l'avenir du produit.

La release à date fixée et à durée uniforme, par exemple une release tous les trois mois, est la formule recommandée.

Pour prendre une métaphore sportive, la release à date fixée s'apparente au test de Cooper. Ce test était pratiqué autrefois à l'armée : il s'agit de parcourir la plus grande distance possible en douze minutes. *A contrario*, la planification de release pour connaître la date de fin se rapprocherait d'une course sur une distance définie, par exemple un 10 km.

16.2. Les bases de la planification de release

16.2.1. Durée fixée, périmètre ajusté

La façon agile de planifier, que nous avons appliquée pour le sprint, renverse le triangle périmètre-coût-délai, qui est même un rectangle si on ajoute la qualité, en considérant

que :

- la date, le coût et la qualité sont définis à l'avance et non négociables ;
- le périmètre (« scope ») constitue la variable d'ajustement.

La planification de release s'appuie sur ce principe, en cherchant à maximiser la valeur dans le choix de ce qui constitue le périmètre.

16.2.2. Vélocité et capacité

La **vélocité** est la mesure de la partie du backlog qui a été réalisée par l'équipe pendant un sprint.

La **capacité** est la prévision de ce que l'équipe devrait être capable de faire pendant un sprint.

Ces deux notions vont nous permettre de planifier. Leur utilisation, qui paraît simple, est parfois une source de gaspillage de temps dans les équipes, en particulier à cause de séances d'estimation longues et sans valeur ajoutée.

16.2.3. Exemple simple de planification de release

Planifier la release, consiste en :

- **Identifier la quantité de travail** pour la durée de la release.
- **Calculer la vélocité moyenne à partir de l'historique** – Il suffit de diviser le total fini par le nombre de sprints effectués.
- **En déduire la capacité** – L'hypothèse la plus simple est de prendre pour capacité la vélocité moyenne.
- **Projeter sur le reste à faire** – En partant de la story la plus prioritaire et allant jusqu'à la moins prioritaire du bac d'affinage, on applique la capacité pour définir le contenu de chaque sprint.

En prenant quelques hypothèses, nous allons montrer que leur utilisation est vraiment simple, tout en donnant des résultats exploitables.

Voici ces hypothèses : les releases ont toutes une durée fixe de 4 mois, elles incluent 8 sprints de 2 semaines, le déploiement auprès des utilisateurs finaux se fait à la fin de chaque release, l'équipe a déjà déroulé une release et collecté des mesures, l'équipe pratique l'affinage et sait bien décomposer en petites stories, on a mesuré qu'une story épique était en moyenne deux fois plus « grosse » qu'une story normale.

Dans notre exemple, nous en sommes au sprint 3. On prend le paquet de stories correspondant à la capacité, c'est la prévision pour le sprint 4. On continue ainsi avec les sprints suivants jusqu'à la fin du bac d'affinage.

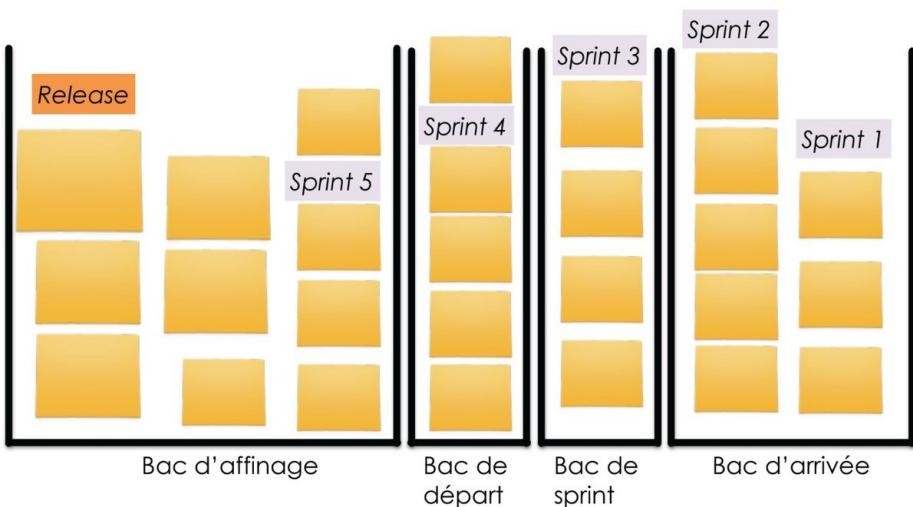


Fig. 16.1 Plan de release correspondant à l'exemple

- **Ajouter les incertitudes** – Toutes les prévisions comportent une part d'incertitude. On en tient compte et on l'indique sur le plan.

Cette approche simple s'appuie sur du découpage, du mesurage et du comptage ; elle ne nécessite pas de faire des estimations poussées.

16.2.4. Quand planifier ?

À part la première fois, il n'y a pas de réunion dédiée à la planification de release. Beaucoup d'activités qui y contribuent se font lors de l'affinage : décomposer, ordonner, estimer, purger.

D'autres activités ont lieu lors de la revue de sprint : mesurer la vélocité, présenter le plan aux parties prenantes.

Bien sûr, si on attend deux ou trois sprints, cela prendra plus de temps qu'une réunion d'affinage habituelle. On pourra insérer une story dédiée à cette longue séance dans le sprint concerné.

16.2.5. Avec qui planifier ?

Toute l'équipe participe à l'activité d'estimation.

L'habitude prise par des managers, avant de passer à Scrum, de faire des plans seuls ou en comité restreint en début de projet peut les pousser à planifier la release sans faire participer l'équipe. En effet, avec la gestion de projet traditionnelle, ce type de plan est élaboré par un ou plusieurs experts du chiffrage munis de leurs abaques. Avec Scrum, c'est différent, les estimations sont faites par l'équipe, ainsi que la planification de release qui en découle.

La suite de la planification de release, après l'estimation, s'effectue éventuellement en comité réduit, avec le Product Owner et un ou deux membres de l'équipe.

Le Product Owner est responsable de la consolidation du plan et de sa communication, entre les activités faites en commun. Il l'actualise à chaque sprint et le présente à la revue.

16.3. Les activités de planification de release

16.3.1. Identifier la quantité de travail pour la release

Le travail à faire est dans le backlog. Si on a mis en place le bac d'affinage et le bac à glace, le périmètre de la release est ajusté facilement par déplacement d'une story entre les deux.

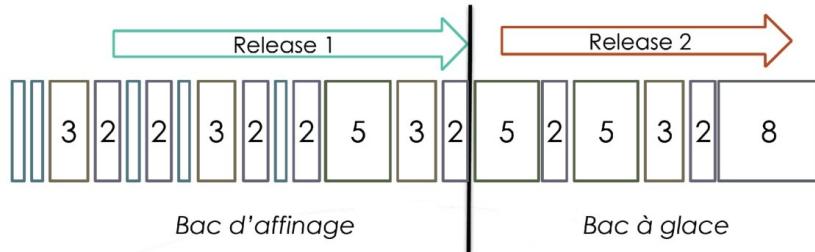


Fig. 16.2 La release à périmètre ajusté

Stories ou features

La planification de release se pratique habituellement avec des stories.

Une alternative est de planifier au niveau des features. Le problème avec un plan basé sur les stories est qu'il risque de ne pas être très éclairant pour les parties prenantes. Les features leur parlent plus. Il suffit que les prévisions soient assez précises pour identifier les points de synchronisation.

Techniques

On peut compter les stories ou les features. Pour être plus précis et tenir compte de la différence entre les éléments, on peut estimer leur taille.

Planning poker

Le planning poker^[1] est devenu très populaire auprès des équipes Scrum. C'est une technique d'estimation en groupe, avec des cartes, qui combine le jugement d'expert et l'estimation par analogie.

Chaque participant reçoit un jeu de cartes. Chaque carte comporte une valeur possible pour l'estimation d'une story.

On utilise généralement une dizaine de valeurs, issues de la suite de Fibonacci : 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Personnellement, je conseille de s'arrêter à 13.

Pour commencer la séance, il faut définir l'étalon. C'est simplement une story connue de tous, pour laquelle l'équipe décide en commun d'une valeur arbitraire.

Il est préférable de choisir une story de taille moyenne et de lui donner une valeur de 3, pour laisser ainsi le spectre ouvert vers le haut et vers le bas.

Déroulement du *planning poker*

- Le Product Owner présente la story.
- Les membres de l'équipe posent des questions pour bien la comprendre et débattent brièvement.
- Tous les participants présentent en même temps la carte qu'ils ont choisie pour l'estimation.
- Le groupe discute des différences éventuelles, en commençant par les extrêmes.
- On fait un deuxième tour de vote pour arriver à un accord par consentement sur l'estimation pour la story, puis on passe à la suivante.

En principe, le PO ne vote pas lors des estimations, qui sont réservées aux développeurs. Cependant, s'il se sent en capacité d'émettre une estimation pertinente, il peut y participer, tout comme les experts qui aideront l'équipe à réaliser des stories.

Quelques leçons tirées des nombreuses sessions de planning poker que j'ai animées : c'est facile à organiser, deux tours de vote suffisent pour converger, la réflexion par analogie est utilisée dans les discussions après le premier vote, c'est fun. En plus de l'aspect estimation, le planning poker favorise une discussion riche entre l'équipe et le Product Owner.

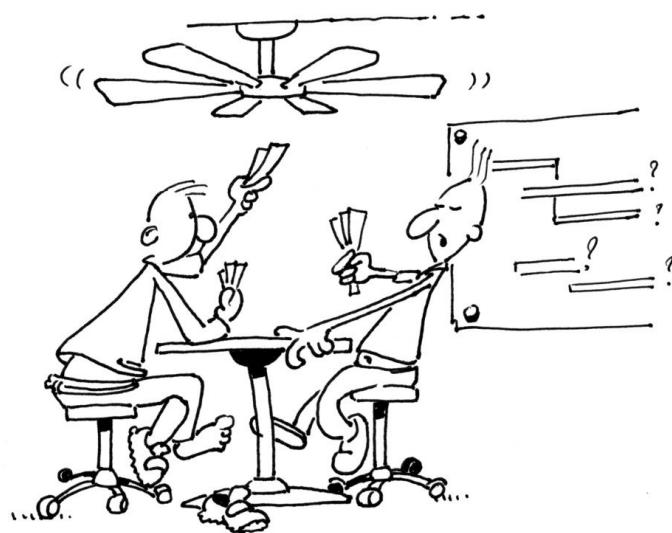


Fig. 16.3 Des mordus du *planning poker*.

Estimation par similitude

La séance initiale de planning poker prend un peu de temps, environ une demi-journée. On peut accélérer le mouvement en pratiquant l'estimation par similitude^[2] (on en parle aussi de « *wall estimation* »).

Cette estimation par comparaison est facilitée par l'usage de Post-it® pour chacune des stories : on fait des rangées pour chaque valeur possible d'une estimation.

Les stories déjà estimées sont visibles, ce qui facilite les estimations des suivantes et permet de raccourcir le temps consacré au planning poker en cas de divergence. Il arrive

également que l'équipe ajuste *a posteriori* une estimation lorsqu'elle la compare à d'autres sur ce tableau.

Bien sûr, cela sera au détriment de la communication mais cela permet de démarrer plus vite.

Taille de T-shirt

La suite de Fibonacci est généralement utilisée avec une dizaine de cartes. C'est beaucoup et cela n'a pas grand sens d'avoir un choix et une échelle aussi importante.

C'est pourquoi nous conseillons de limiter le nombre de valeurs.

Une technique plus simple est d'utiliser les tailles de T-shirts. Trois tailles suffisent : S, M, L. Cette façon de procéder a l'avantage de ne pas porter sur des valeurs numériques, ce qui facilite l'estimation relative sans arrière-pensée. On pourra leur donner des valeurs numériques, après l'estimation collective, pour faciliter les calculs.

Comptage

La technique simple que nous avons utilisée en introduction revient presque à compter. Il faut d'abord identifier si un élément est une épique ou une story. Ensuite, il suffit de compter dans chaque groupe. L'hypothèse « une épique égale deux stories » permet d'obtenir la quantité de travail à fournir, sans passer du temps à estimer.

16.3.2. Mesurer la vitesse et déduire la capacité

Selon qu'on le fait avec des stories ou des features, qu'on estime en points ou qu'on compte, cela fait plusieurs possibilités pour mesurer la vitesse d'un sprint :

- **En nombre de stories** – Pour mesurer la vitesse d'un sprint, on peut simplement compter les stories finies. Comme elles sont passées par la définition de prêt, qui en général inclut une vérification sur la taille, on peut considérer que leur variabilité est statistiquement faible. Cette hypothèse doit bien entendu être vérifiée avec les mesures faites à chaque sprint.
- **En nombre de features** – L'avantage avec les features est qu'elles varient probablement moins en taille que les stories, ce qui rend le comptage moins aléatoire et permet d'aller plus vite. Cependant, cela dépend du contexte et de l'expérience des participants.
- **En points** – La vitesse s'obtient en totalisant les points des stories (ou features) finies.

Pour prévoir la capacité, on se base sur la vitesse des sprints passés.

Dans l'exemple de la figure 16.1, nous sommes en train de finir le sprint 3. Pendant ce sprint, nous aurons fini 4 éléments (c'est juste l'exemple du schéma, en général une équipe en trait plutôt 10). Cela correspond à la moyenne observée dans les deux sprints passés, plus tous ceux de la release précédente.

C'est le même principe avec des features : la différence est que le nombre de features terminées par sprint est bien plus réduit. Il faut attendre un peu plus longtemps pour

obtenir des données significatives utilisables pour la planification.

Exemple : on a mesuré une moyenne de 2 features finies par sprint, il reste 5 sprints dans la release, on prévoit donc d'en terminer 10 de plus d'ici la fin.

Comme pour le comptage des stories, cette simplification n'est possible que si l'équipe possède une expérience poussée dans le découpage homogène.

Quand a-t-on assez de données historiques ?

Comme nous l'avons vu, la planification est basée sur la capacité de l'équipe, qui est elle-même dérivée de la vélocité mesurée en fin de sprint. Il est donc nécessaire d'avoir des mesures pour planifier la release. Soit l'équipe a effectué des sprints dans une release précédente, soit il faut attendre quelques sprints dans cette release pour en collecter.

Un ou deux sprints mesurés, ce n'est pas assez, trois ou quatre cela commence à donner des prévisions plus réalistes pour la capacité.

Pour la première release d'un produit, on peut donc commencer la planification après quelques sprints. Il sera toujours possible de prendre en compte la taille des stories déjà finies pour mesurer la vélocité si on la calcule en points. Pour les releases suivantes, si l'équipe reste stable, nous aurons déjà des chiffres, et le plan de release pourra être élaboré dès le sprint zéro.

16.3.3. Projeter sur le reste à faire

Le backlog étant ordonné, cette dernière activité est un jeu d'enfant pour qui sait compter.

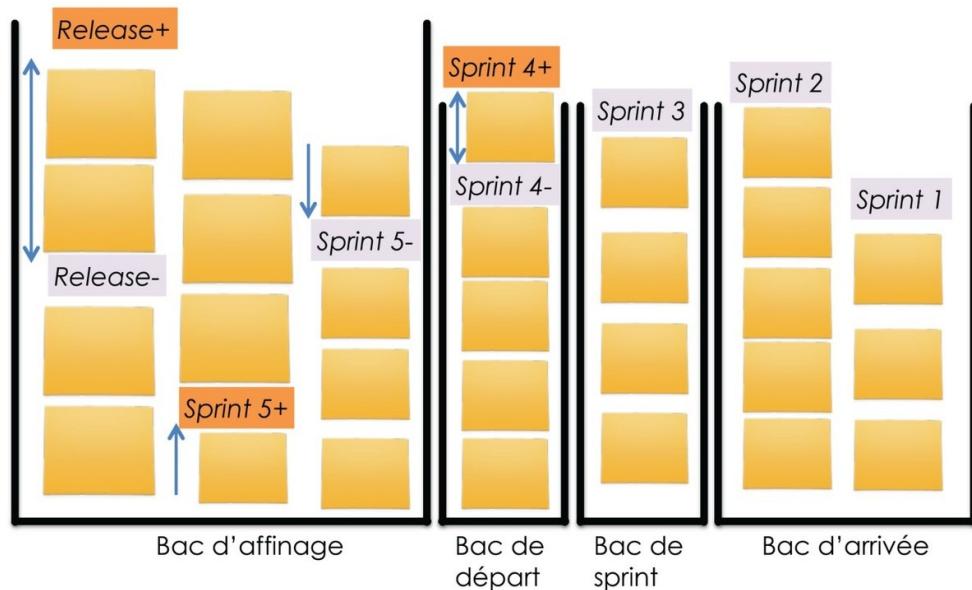


Fig. 16.4 Plan de release avec incertitudes

Exemple : pour les prévisions des sprints suivants, nous utilisons la moyenne des sprints passés, soit 4 éléments par sprint. Il y a 5 éléments dans le bac de départ, nous ajoutons l'indication de fin de sprint 4 après le 4^e élément. Nous continuons dans le bac d'affinage pour le sprint 5.

Dans ce bac, il y a aussi des plus gros éléments. Notre hypothèse est qu'ils sont deux fois plus gros. Pour projeter la capacité sur les bacs, on **compte** un point par story normale et deux points par story épique.

Sachant qu'il y a 8 sprints dans la release, le bac d'affinage est rempli avec la prévision de ce qu'on pense finir dans les 8 sprints.

16.3.4. Prendre en compte les incertitudes

Les hypothèses sur la taille des stories comportent une part d'incertitude. On ne prend pas de marge pour chaque élément, mais pour l'ensemble. La marge s'applique sur un périmètre, qu'on peut montrer sur chaque sprint et sur la release.

Quelle marge prendre ? Je suggère, à partir du moment où on a déroulé quelques sprints, de prendre l'écart-type de la vitesse.

Pour apporter de la visibilité sans que cela soit difficile à comprendre, il suffit de montrer la marge sur deux sprints à venir et sur la release.

La façon de montrer cette marge d'incertitude varie (figure 16.4 et 16.5).

La mesure sur les sprints passés nous permet de prendre l'hypothèse suivante : nous allons réaliser 4 à 5 éléments par sprint. Nous avons des éléments prêts dans le bac de départ sur lesquels appliquer cette prévision.

Dans un bac, l'élément en bas à droite est au premier rang, celui au-dessus au deuxième, celui en haut à gauche au dernier. L'indication Release- constitue la prévision basse, le minimum de ce qui sera fini. L'indication Release+ représente la prévision haute, le maximum qu'on puisse finir. Il n'est pas utile de surcharger la présentation avec les indications pour les sprints éloignés (sprints 6 et 7 ici).

Cette présentation avec incertitudes permet de :

- s'assurer que tout ce qui est essentiel pour la release est bien inclus dans la prévision basse ;
- communiquer des prévisions facilement compréhensibles aux parties prenantes.

Plus on avance dans la release, plus l'incertitude diminue car on dispose de davantage de données réelles, *a priori* plus convergentes.

Ce plan de release avec incertitudes met en évidence une partie du backlog qui sera finie à coup sûr, une autre qui ne le sera sûrement pas, et entre les deux, la zone d'incertitude.

Je conseille de déplacer les stories qui ne seront sûrement pas finies, en tenant compte de la marge d'incertitude, dans le bac à glace, dès que possible (lors de la purge pendant l'affinage). Comme cela, le bac d'affinage est toujours à jour et les indicateurs seront plus faciles à réaliser et à interpréter.

16.4. Engagement sur le plan de release

De la même façon qu'une équipe s'engage lors de la planification du sprint, doit-elle s'engager sur un résultat au début de celle-ci ?

16.4.1. Engagement de résultat ?

Tout d'abord, il ne faut pas confondre prévision et engagement. Un plan de release présente des prévisions et il peut, éventuellement, permettre de prendre un engagement.

Ensuite, l'horizon étant plus lointain, un engagement doit prendre en compte les risques liés à l'incertitude.

Bien sûr, un engagement sur des tâches n'a pas de sens. De même, il n'est pas sérieux de s'engager sur des stories, puisque la plupart de celles qu'on va finir dans la release ne sont pas connues au début.

Nous savons qu'il faut laisser une variable d'ajustement et que celle-ci est le périmètre. Un engagement qui serait pris sur une liste de features laisse de la marge de manœuvre sur les stories qui les composent. Cependant, il n'est envisageable que si l'équipe possède assez de données historiques et si la taille des features est maîtrisée.

16.4.2. Objectif de la release

En général, il n'y aura donc pas d'engagement sur un périmètre. Et sur l'objectif de la release ?

L'objectif de la release est une notion similaire à l'objectif du sprint, à plus haut niveau. Il est préparé en avance, consolidé ensuite et adopté lorsque la release commence.

L'impact mapping présenté dans le chapitre 14 *Découvrir le produit* est une technique permettant de converger collectivement vers l'objectif de la release. Il apparaît au centre de la carte et peut être accompagné d'éléments chiffrés vérifiables.

Exemple : pour la release Néouvielle de Peetic, l'objectif est d'avoir gagné 1 M€ grâce aux abonnements Premium.

Bien entendu, comme pour l'objectif de sprint, celui de release sera revu si les conditions initiales changent.

16.4.3. Engagement contractuel

La question revient régulièrement de la compatibilité de l'agilité avec un « contrat au forfait ». Notre réponse, simple, découle de ce que nous venons de présenter : sans données historiques fiables, un engagement sur un résultat n'a pas de sens.

Un engagement sur une vélocité, comme on en voit parfois dans les contrats, n'est pas plus recommandé.

Moyennant les conditions présentées sur l'historique, un engagement sur une liste de features est envisageable. Une condition supplémentaire porte sur le PO, qui devra effectivement jouer son rôle, à savoir définir les features minimales apportant de la valeur.

Il est possible de définir un budget par feature, et de le réaffecter entre features tout en restant dans l'enveloppe globale, c'est le reflet de l'ouverture au changement de l'agilité.

16.5. Résultats de la planification de release

16.5.1. Plus de transparence

Le premier résultat est une équipe avec une meilleure connaissance de ce qui l'attend dans les sprints qui viennent. Les parties prenantes trouveront avec le plan de release des informations qui les intéressent sur les résultats attendus dans le futur. La transparence est renforcée.

16.5.2. Le plan de release

Un plan de release présente les sprints à venir et le contenu prévu de ces sprints, qui est défini par les stories ou les features associées, selon le cas.

Il est mis à jour régulièrement : à la différence d'un plan détaillé fait à l'avance qui devient la référence intouchable, le plan de release évolue pour tenir compte des changements.

Représentations du plan de release

Nous avons montré le plan de release « dans les bacs ». Cela permet d'avoir tout au même endroit, d'éviter de dupliquer les stories, c'est plus facile.

En revanche, cela peut dérouter ceux habitués à voir l'horizon le plus lointain sur la droite.

Une alternative est de présenter le plan de release à part, sous forme de tableau. Les sprints sont présentés de façon séquentielle de gauche à droite. L'important est qu'il soit bien visible : les informations contenues dans le plan de release servent à anticiper les interdépendances.

Il n'est pas utile d'être aussi précis sur ce qui vient à court terme que sur ce qui arrivera dans quelques sprints. Cela peut être reflété dans le plan de release, en ne montrant que ces deux horizons.

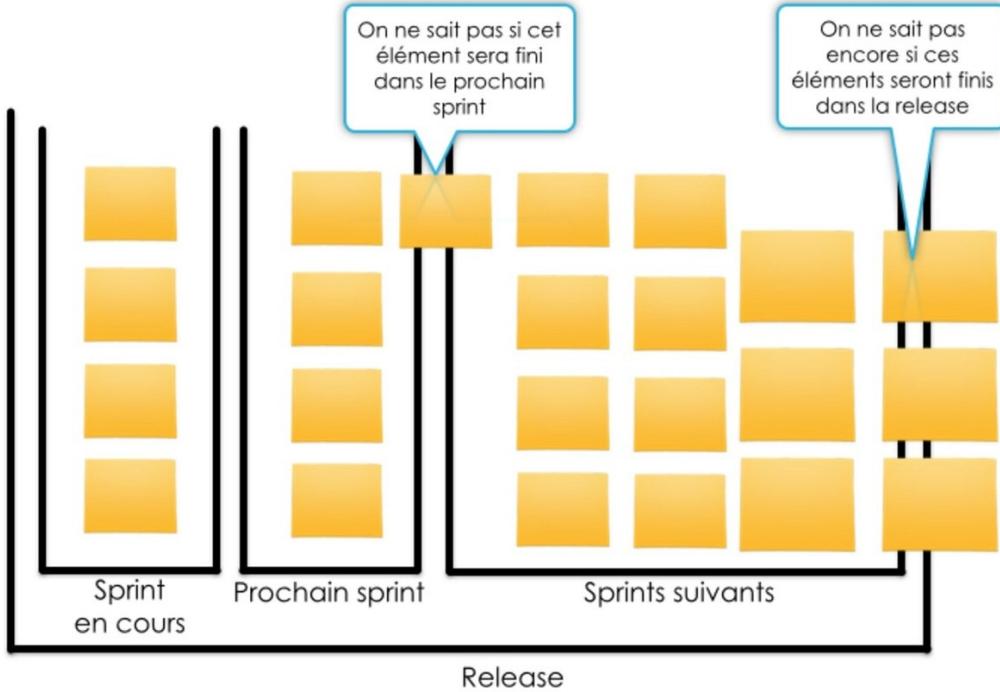


Fig. 16.5 Un plan de release à deux horizons, avec incertitudes

On a ajouté les incertitudes en mettant les stories incertaines à cheval sur le bord de l'horizon concerné.

Une autre façon de montrer les incertitudes dans le plan est de présenter le backlog en trois zones (figure 16.6) : ce qu'on est sûr de finir compte tenu de la vitesse maximale, ce qu'on est certain de ne pas faire (vitesse basse), et entre les deux la zone d'incertitude.

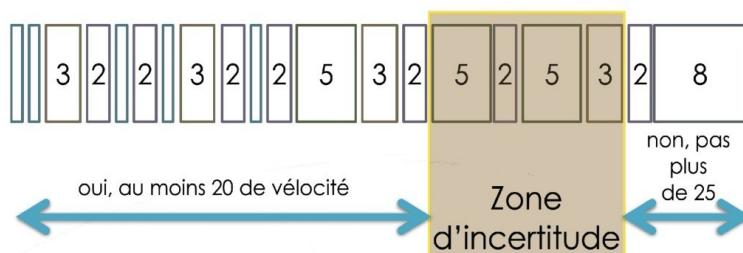


Fig. 16.6 Un backlog avec zone d'incertitude

16.6. La planification de release sur le terrain

16.6.1. Attention à ne pas prévoir sans données historiques

La vitesse est une mesure : il faut que l'équipe ait vécu une expérience commune pour l'avoir collectée. Quand un développement commence et qu'une nouvelle équipe vient d'être constituée, elle n'a pas de passé, elle n'a donc pas de vitesse connue. Si on veut quand même faire un plan de release, sans attendre de dérouler un ou deux sprints, il faudrait investir sur une estimation de sa capacité.

Pas de miracle, l'agilité ne permet pas de deviner avec précision un avenir incertain.

16.6.2. Relativiser l'importance de la vitesse

Des gestionnaires découvrant l'agilité se jettent sur la vitesse : ah, des chiffres, on va pouvoir mesurer, contrôler, « challenger », etc. On va comparer des équipes, comparer les gens, etc.

Pas la peine d'en rajouter, vous pouvez vous faire une idée des dérives constatées.

La vitesse est une mesure de l'équipe, pas de personnes individuelles.

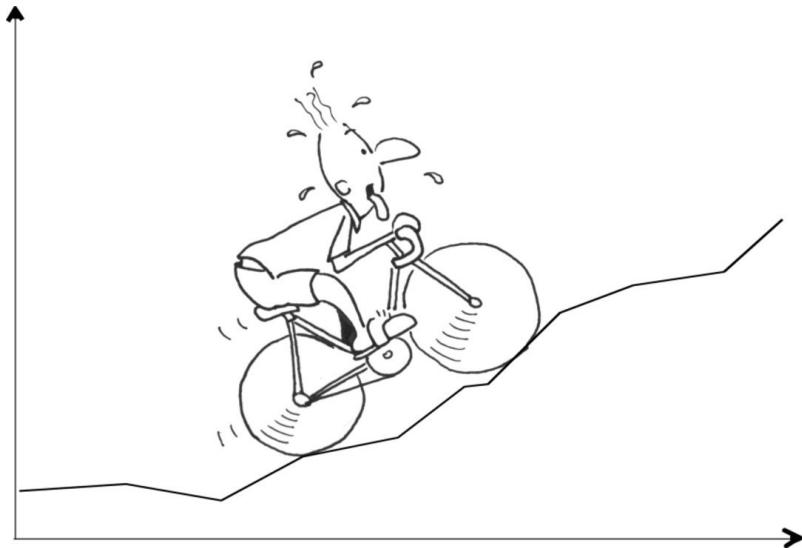


Fig. 16.7 On ne mesure pas la vitesse individuelle

16.6.3. Arrêter quand le backlog est vide ?

Ceux qui sont habitués à développer à partir d'une spécification contenant tout ce qu'il y a à faire seront tentés de dire que la release s'arrêtera quand tout le backlog sera vide. Mais le backlog est vivant et, finalement, il se rapproche d'un flot continu toujours alimenté.

Ce n'est donc pas une bonne idée de vouloir vider le backlog. J'ai connu une équipe qui a essayé pendant 18 sprints, sans succès !

L'utilisation du bac d'affinage permet d'aller plus loin. Il contient la sélection du sous-ensemble du backlog pour la release. Il suffit alors de dire : « *on s'arrêtera pour la release courante quand le bac sera vide* », à condition de le purger régulièrement.

C'est ce qu'on appelle une release à périmètre fixé. La planification de la release a alors pour objectif de prévoir la date de fin.

Même restreint de cette façon, le périmètre d'une release peut toujours évoluer, il serait « balot » de figer le bac d'affinage en refusant un changement qui apporte de la valeur.

16.6.4. Planifier plusieurs releases

Un plan de release présente les prévisions sur la durée des sprints, sur quelques mois. On peut commencer le plan de release $n + 1$ alors qu'on est encore dans la release n . Une solution simple est de déplacer dans le bac à glace le contenu de la release $n + 1$.

Ensuite, les stories y sont ordonnées et on peut reprendre les techniques que nous avons appliquées au bac d'affinage pour planifier cette prochaine release, au moins les premiers sprints.

16.6.5. S'adapter au calendrier

La pertinence de la vélocité repose sur une durée fixe des sprints associée à une composition de l'équipe qui ne change pas d'un sprint à l'autre.

La planification doit tenir compte des événements connus à l'avance, comme les ponts en mai ou les vacances en été, qui peuvent réduire la capacité d'une équipe.

L'équipe Peetic fait habituellement des sprints de deux semaines. Elle commence un nouveau sprint et les membres de l'équipe font les ponts de mai. Pour avoir à peu près la même capacité que pour les autres sprints, il est décidé de passer la durée de ce sprint à trois semaines. Cela aurait pu être anticipé dans le plan de la release.

La durée d'un sprint peut exceptionnellement varier pour garder la capacité à peu près stable.

Si la taille de l'équipe évolue pendant la release, la mesure de la vélocité est évidemment moins fiable : une personne en plus ou en moins, cela a un impact sur sa capacité, en particulier pour les petites équipes. Un autre inconvénient est d'ordre collectif : un nouvel arrivant doit apprendre à s'intégrer dans l'équipe, cela lui demande du temps et l'équipe y consacre aussi du temps.

16.6.6. Répondre aux questions essentielles

Il vaut mieux être globalement juste que précisément faux. Le plan de release est là pour répondre à des questions essentielles, pas pour faire du micro-management.

Revenons à nos questions du début à propos de Peetic :

– **Pourrons-nous utiliser la gestion des inscriptions en ligne pour le salon du chien en mars ?**

Oui. Le plan de release est présenté avec des features et celle-ci fait partie de celles que l'équipe est sûre de faire.

– **Dans combien de temps aurons-nous la possibilité d'utiliser l'application sur tablette ?**

Cette feature est prévue dans la release suivante, qui se termine en août.

– **Quel est le budget nécessaire pour développer la release 2 ?**

Peetic a choisi la release à date fixe. Le budget est fixé à l'avance, on l'obtient facilement avec la taille de l'équipe et la durée.

– Quand le composant de paiement en ligne est-il nécessaire ?

Dans deux sprints.

Bien commencer

La question à se poser

Est-ce que nous avons vraiment besoin de prévoir à moyen terme ? Sur quel horizon ? Avec quelle précision ?

L'équipe a l'impression de passer beaucoup de temps à faire des estimations.

De mauvais signes

Le plan de release n'est pas expliqué aux parties prenantes.

La vitesse est utilisée pour mettre la pression sur l'équipe.

Par quoi démarrer

Mettre en place les bacs, cela rend plus facile la présentation du plan de release.

Une lecture pour tous

No Estimates, [Duarte, NoEstimates], un livre (pour une fois en anglais) au titre provocateur, mais qui pose de bonnes questions sur l'intérêt des estimations.

À retenir

Une caractéristique importante des méthodes agiles est leur capacité à prendre en compte les changements. Cela implique que les plans sont remis à jour régulièrement. C'est particulièrement vrai pour le plan de release, qui est actualisé à chaque sprint.

Cette adaptation au changement s'accompagne d'anticipation : le plan de release permet de prendre des décisions sur le produit.

Pour planifier la release, il faut identifier la quantité de travail à faire, mesurer la vitesse pour en déduire la capacité prévue, projeter sur le reste à faire et ajouter la marge d'incertitude. Il est important de réfléchir à la précision souhaitée pour éviter de gaspiller du temps à faire des estimations.

Références :

☞ Mike Cohn, *Agile Estimating and Planning*, Prentice Hall, 2005.

☞ Vasco Duarte, *NoEstimates*, 2014.

<http://noestimatesbook.com/>

Notes

- [1] Pour plus d'informations, lire : <http://www.planningpoker.com/detail.html>.
- [2] Pour en savoir plus : <http://www.aubryconseil.com/post/L-estimation-par-similitude-d-effort>.

Tirer profit des outils

En France, on a sûrement trop tendance à faire passer le choix de l'outil avant l'appropriation de la méthode par ceux qui sont censés l'appliquer. Je me souviens d'une grande société dans le domaine aéronautique où des armées de développeurs passaient leur temps à outiller une méthode alors que celle-ci n'était pas encore maîtrisée. Sur cet aspect, les méthodes agiles prônent le bon sens qui consiste à faire passer l'acquisition des valeurs et des pratiques avant le choix d'un outil.

Certains pourraient aller trop loin en interprétant le premier énoncé du *Manifeste agile*, « *les gens et leurs interactions ont plus de valeur que les outils et les processus* », comme une recommandation de ne pas utiliser d'outils pour le développement agile.

Ce qui compte en premier, ce sont les gens, mais il faut évidemment des outils pour développer ; par exemple, les pratiques d'intégration continue et de pilotage par les tests nécessitent les outils adéquats.

En ce qui concerne les pratiques Scrum, le besoin d'outil dépend du contexte : bien sûr, dans le cas d'équipes regroupées dans le même espace, il est moins fort que dans le cas d'équipes réparties sur plusieurs continents.

Le plus intéressant, c'est que les méthodes agiles revisitent la notion d'outil. Il n'y a pas que des outils informatiques. L'informatique n'est qu'une solution qui vient avec son lot de fonctionnalités. Mais, et c'est d'ailleurs ce que nous enseigne l'*impact mapping*, il vaut mieux définir le problème avant de s'occuper de la solution.

L'objectif de ce chapitre est de présenter des outils qui aident une équipe Scrum : des applications informatiques, mais aussi des jeux, des tableaux physiques et, bien sûr, des Post-it®.

17.1. Les Post-it®

Dans une entreprise, on reconnaît l'utilisation de Scrum à la présence de notes de couleur sur les murs des bureaux.

Avant l'avènement du Post-it®, on utilisait des cartes, en fait des fiches bristol. Cet usage était courant dans les premières expériences qu'ont faites les équipes avec les *user stories*. Le C de carte est le premier des « 3C », article fondateur de Ron Jeffries.

J'ai utilisé des cartes pour mes premiers backlogs avec Scrum. C'est pratique pour classer les *stories* par priorité quand on est Product Owner, ainsi que pour écrire des détails au fil des réunions, au verso. C'est plus difficile à partager, sauf à les accrocher sur un tableau en liège ou à les coller au mur.

Pour coller, autant choisir des notes adhésives qui se déplacent facilement : les Post-it®.

Quand une équipe dispose d'un espace de travail ouvert ou simplement d'un mur, c'est l'outil le plus efficace pour communiquer efficacement entre tous les membres.

L'utilisation de Post-it® est particulièrement recommandée pour le suivi d'un sprint. Le tableau Scrum est l'endroit idéal devant lequel tenir la mêlée quotidienne.



Fig. 17.1 Post-it® war

Le Post-it® se colle facilement et permet de jouer avec les couleurs. En revanche, par rapport à la carte, il n'a qu'une face facilement utilisable. Cela pousse à être concis, d'autant plus qu'il faut écrire gros pour que ça se voie sans s'approcher à quelques centimètres du mur.

Et puis le Post-it® peut s'envoler, ce qui est fâcheux pour la pérennité des informations qu'il contient. À Toulouse, le vent d'Autan qui souffle par rafales est l'ennemi des tableaux avec des Post-it®. Je me souviens d'une ouverture de fenêtre après une réunion de planification de sprint qui a entraîné une chasse aux Post-it®, avec l'équipe à quatre pattes pour les récupérer.

Les Post-it® permettent également de faciliter le travail collectif et créatif lors des différents ateliers permettant de constituer le backlog. Dans ce cas, la pérennité n'est pas nécessaire au-delà de la réunion.



Fig. 17.2 Product Owner tentant de capitaliser dans un backlog après un atelier avec Post-it®.

Mon conseil est d'utiliser toujours des Post-it® pour les ateliers et, chaque fois que c'est possible, pour le tableau de sprint.

C'est la base du management visuel, vecteur de transparence.

17.2. Les outils informatiques

17.2.1. Les tableurs

Selon les enquêtes, l'outil informatique le plus utilisé sur les projets agiles reste Excel. Essentiellement pour le backlog : les stories sont les lignes et leurs attributs sont des colonnes.

Je trouve beaucoup d'inconvénients à l'utilisation d'un tableur. Le plus significatif est que cet outil ne favorise pas le partage entre les personnes de l'équipe.

Un tableur en ligne permet de limiter cet inconvénient.

17.2.2. Les bugtrackers

Les outils permettant de gérer les bugs sont devenus populaires. Ils sont basés sur des tickets, chacun suivant un *workflow* permettant de tracer la vie du bug jusqu'à sa correction. Certaines équipes qui utilisaient des *bugtrackers* avant de passer à Scrum continuent à s'en servir, en essayant de faire rentrer Scrum dans le *workflow* proposé.

Le risque est grand de rester dans la logique ancienne en mélangeant des concepts différents : un ticket pour une tâche, pour une story ?

17.2.3. Les outils agiles

Il existe de nombreux outils dédiés à Scrum et notamment à la gestion du backlog de produit. En cherchant un peu, on en trouve plusieurs dizaines, c'est un sujet qui semble inspirer les innovateurs. Le ticket d'entrée, comme on dit, n'est pas très élevé pour réaliser un outil qui gère simplement un backlog ou possède trois colonnes pour le sprint.

Cependant, un backlog n'est pas une simple *todo list*.

Il existe des outils commerciaux et des outils libres, des outils fonctionnant selon différents modes, avec différentes technologies.

J'ai participé activement au développement de l'un d'eux, iceScrum, outil open source. C'est parti d'une proposition que j'avais faite pour un projet d'étudiants en 2005. J'ai tenu le rôle de Product Owner d'iceScrum pendant des années. Selon le principe *Eat your own dog food*, j'ai longtemps utilisé iceScrum, presque tous les jours. J'ai rencontré de très nombreux utilisateurs d'iceScrum ou d'autres outils et j'ai écouté leurs demandes. J'ai constaté bien des dérives : en premier, moins de conversations, mais aussi des détournements de l'outil vers plus de centralisation et plus de contrôle, et du temps perdu à de la collecte de données sans intérêt.

J'ai beaucoup utilisé iceScrum mais aussi des tableurs, des bugtrackers et d'autres outils. La plupart du temps, j'ai constaté cette tendance :

L'outil informatique réduit, voire détruit, la conversation, dont on a vu l'importance pour l'affinage du backlog.

17.2.4. Cohabitation Post-it® et outil informatique

Contrairement à une idée répandue, on n'a pas à choisir entre outil informatique et Post-it®, on peut utiliser les deux^[1].

Pour les équipes qui sont co-localisées, l'usage d'un outil pour le plan de sprint n'est pas recommandé : un tableau des tâches avec les Post-it® est plus efficace.

Si l'équipe est physiquement dans le même espace, il y a tout intérêt à utiliser des Post-it® pour le plan de sprint. Cela présente de nombreux avantages, comme l'appropriation des tâches, le meilleur suivi visuel, etc.

Si on utilise plusieurs outils, il vaut mieux que l'information ne soit qu'à un seul endroit et éviter les saisies en double, par exemple les tâches sur des Post-it® et aussi dans un outil. En faisant cohabiter un outil pour le backlog et des Post-it® pour le sprint, la redondance est très limitée.

17.3. Les tableaux

Le tableau est un outil significatif pour une équipe qui fait du Scrum. Il contribue au management visuel.

17.3.1. Tableau pour le plan de sprint

Le tableau Scrum classique, c'est celui pour le plan de sprint. Il se présente avec une colonne à gauche pour les stories en sprint et trois colonnes pour les tâches.

Des astuces pour un joli tableau

En côtoyant les équipes Scrum, on s'aperçoit que l'aspect esthétique est important. Les équipes qui réussissent le mieux ont généralement de jolis tableaux. Cela facilite la transparence : les informations sont facilement compréhensibles.

Voici quelques astuces pour y parvenir :

- Éviter le Post-it® qui rebique. En le décollant délicatement à partir d'un côté et non de bas en haut^[2].
- Utiliser un fil déplaçable pour les colonnes. Eh oui, on se rend vite compte que les colonnes à largeur fixe sont un problème. Une de mes équipes a utilisé du fil électrique scotché en haut et en bas, et donc facilement déplaçable. On peut aussi se servir de laine ou de ruban adhésif, celui utilisé habituellement pour la protection quand on fait de la peinture.
- Jouer sur les couleurs. On peut utiliser les couleurs pour plein d'usages, et ce n'est pas anecdotique^[3].
- Jouer avec les tailles des Post-it®. Souvent, les Post-it® pour les tâches sont plus petits que ceux pour les stories. J'ai vu des équipes utiliser la taille du Post-it® comme indicateur de celle de la tâche.
- Se servir de la position du Post-it®. Les colonnes sont en général plus larges que le Post-it®, ce qui permet de jouer sur la position pour montrer des choses subtiles comme l'avancement de la tâche. On peut aussi s'en servir comme aide-mémoire : en mettant une tâche qui vient de se finir très à gauche de la colonne Fini, on se souviendra lors de la mêlée quotidienne qu'elle a été finie depuis la précédente réunion. Certains mettent des Post-it® à cheval sur deux colonnes.
- Utiliser des magnets avec avatars. Pour montrer qui a pris la tâche, l'usage habituel est de mettre ses initiales sur le Post-it®. Un avatar représentant un développeur sera plus visuel, en plus d'être amusant. L'autre intérêt est de limiter le multitâches en ne faisant qu'un seul ou deux magnets par personne.
- Tamponner ou pencher les stories finies. Sur un tableau, il n'y a souvent qu'une colonne pour les stories. Pour distinguer celles qui sont finies des autres, si on n'a pas de bac d'arrivée, l'équipe définit une représentation visuelle, comme la pencher, la retourner ou la tamponner avec un gros FINI.

Bannir la colonne supplémentaire

Le tableau Scrum se présente avec quatre colonnes. Des équipes, soit parce qu'elles ont du mal à finir les stories, soit parce qu'elles ont entendu parler de Kanban, sont tentées d'ajouter d'autres colonnes, les plus typiques étant « à vérifier » ou « à tester ».

Si on fait du Scrum et si on veut le faire bien, je déconseille tout à fait cette pratique. La colonne supplémentaire est le signe que l'équipe n'est pas pluridisciplinaire. Ajouter une colonne « à tester », par exemple, montre que le testeur n'est pas proche des développeurs, puisqu'on est obligé de créer une file d'attente pour le test. Cela a tendance à amplifier le multitâches, et le changement de contexte prend du temps.

Avec une ou plusieurs colonnes en plus, le tableau devient plus difficile à lire. En outre, les gens à qui je pose la question ne savent pas si la colonne supplémentaire s'applique aux stories ou aux tâches.

Une des raisons de l'essaimage, présenté dans le chapitre 9 *Planifier le sprint*, est d'éviter cette dérive.

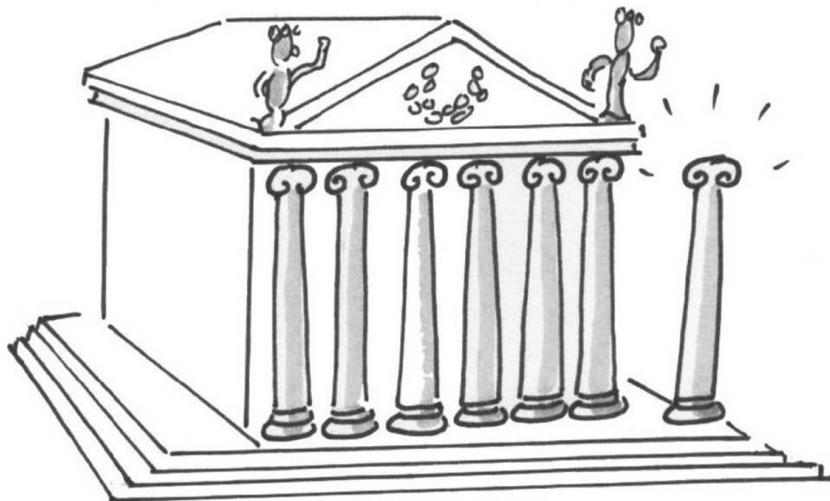


Fig. 17.3 Avec Scrum, on n'a pas besoin d'ajouter une colonne supplémentaire.

17.3.2. Les autres tableaux

Tableau des obstacles

Un obstacle empêche le travail d'avancer comme souhaité. Pour le montrer, une première chose à faire est de mettre en évidence ce qui est freiné ou bloqué. En général, ce sont des tâches et on l'indique sur le tableau Scrum en collant une pastille rouge sur la tâche concernée.

L'étape suivante est de visualiser les obstacles dans un tableau sur trois états, comme les tâches, mais adapté : « à traiter », « en cours de résolution », « éliminé ».

Je conseille d'ajouter la date d'apparition de l'obstacle sur le Post-it®.

L'élimination des obstacles ne doit pas traîner en longueur. La date est aussi utile pour la rétrospective, quand on établit la *timeline* du sprint, on pose les obstacles sur la chronologie des événements majeurs du sprint.

Tableau pour les stories et les features

Comme on l'a vu dans les chapitres sur le backlog, une story passe d'un bac à l'autre : du bac à sable au bac d'affinage, etc.

Les bacs sont pratiques car ils regroupent des stories sur lesquelles on peut faire le même type d'actions. Les bacs côte à côte équivalent à un tableau et présentent la vue générale.

Les deux bacs dans lesquels il y a du travail significatif (des activités) sont le bac d'affinage et le sprint. On peut leur associer une définition de prêt et de fini, placée au-dessus, en dessous, ou à cheval.

Un tableau avec les features donne une vue de plus haut niveau, qui va intéresser les parties prenantes. Le nombre de colonnes dépend du contexte. On peut ajouter, sur le tableau, les règles permettant le passage d'une colonne à la suivante.

Tableau à plusieurs niveaux

On peut représenter deux niveaux sur un seul tableau. Exemple : on élargit le sprint pour ajouter les trois colonnes des tâches.

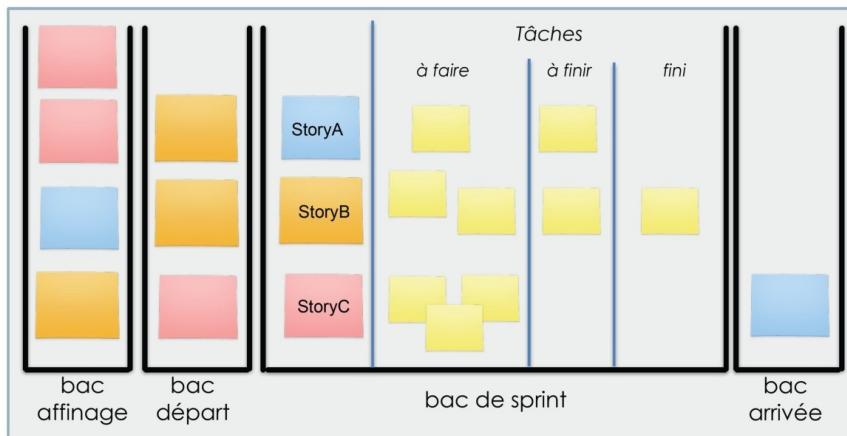


Fig. 17.4 Tableau (Scrum !) à deux niveaux : story et tâche.

17.3.3. À côté du tableau

Le management visuel ne se limite pas au suivi avec des tableaux. À côté, sur le mur, on peut faire figurer des informations pour l'équipe, ou destinées aux parties prenantes.

Il est intéressant de voir, à côté du tableau, l'objectif du sprint, les définitions de prêt et de fini, les résultats de la rétrospective.

Éventuellement, on ajoutera un calendrier visuel mettant en évidence les disponibilités des experts ou les vacances des développeurs.

Un *niko niko*^[4], placé sur la porte de sortie, permettra de se faire une idée de la satisfaction des membres de l'équipe jour après jour.

Les parties prenantes seront intéressées par voir, à côté du tableau de *features*, une *roadmap*, le plan de release et un indicateur comme le *burndown* de release.

17.4. Les outils et les tableaux sur le terrain

17.4.1. Attention à ne pas utiliser en solo

Le risque avec un outil informatique (il est moindre avec les Post-it[®]) est qu'il soit utilisé par une seule personne et non partagé avec toute l'équipe. Par exemple, et en particulier

avec ceux qui font de l'Excel, il n'y a que le Product Owner qui travaille sur le backlog. Un autre exemple est celui du ScrumMaster qui met à jour les tâches de toute l'équipe dans un plan de sprint. C'est évidemment à éviter : le backlog, et encore plus le plan de sprint, appartiennent à l'équipe tout entière.

17.4.2. Éviter de surcharger un tableau

Certains soignent l'esthétique des tableaux et c'est bien. D'autres se disent que le plus important n'est pas qu'il soit beau mais impressionnant.

On m'en a montré de fort impressionnantes. Effectivement, la quantité d'éléments dans les nombreuses colonnes était marquante. C'est souvent au détriment de la simplicité d'utilisation et à l'opposé de la recommandation de limiter le stock.

Il ne faut pas essayer de tout mettre dans un seul tableau. Nous avons vu qu'il y a trois niveaux d'éléments dans Scrum : feature, story, tâche. Cela mérite au moins deux tableaux différents.

17.4.3. Fuir les usines à gaz

Si le besoin d'intégration des outils est réel, l'outil qui fait tout est dangereux. En particulier, l'ajout de fonctionnalités agiles à des outils conçus dans un autre objectif risque d'apporter une complexité importante et de pousser à un usage dévié des pratiques agiles.

Dans un article sur les outils agiles^[5], Kent Beck pense qu'il est préférable d'investir dans un (ou plusieurs) outil rudimentaire adapté à notre façon de travailler plutôt que d'essayer de s'adapter à un (gros) outil plus sophistiqué mais prévu pour une autre approche du travail.

17.4.4. Privilégier les outils open source

Le mouvement du logiciel libre véhicule des valeurs communes avec celles du mouvement agile. Dans les deux cas, on met en avant la notion de partage. Les méthodes agiles sont fréquemment utilisées pour les développements par des communautés open source. C'est pourquoi je recommande de s'intéresser d'abord aux outils qui proposent une licence open source.

17.5. Les jeux

Le jeu est aussi un outil pour une équipe Scrum. Il peut être utilisé pour apprendre mais aussi pour faire le travail, c'est-à-dire aider à produire des résultats.

Faire le travail en jouant, voilà un bel oxymore ? Et pourtant, dans *La troisième révolution industrielle*, Jeremy Rifkin écrit [Rifkin, p. 378] :

Aux XIX^e et XX^e siècles, être industriel était la qualité première de l'homme et devenir un travailleur productif le but de sa vie. Des générations d'humains ont été transformées en machines dans une

inlassable quête de la richesse matérielle : nous vivions pour travailler. La troisième révolution industrielle et l'ère coopérative offrent à l'humanité l'occasion de s'affranchir, de desserrer l'étreinte d'une vie mécanisée bien au chaud dans un monde utilitariste, et de respirer avec ivresse l'air de la liberté : nous vivons pour jouer. Le philosophe français Jean-Paul Sartre a bien vu l'étroite parenté entre jeu et liberté : « Dès qu'un homme se saisit comme libre et veut user de sa liberté », écrit-il, « ... son activité est le jeu. » À quoi je n'ajouterais qu'une seule question : se sent-on jamais plus libre qu'en jouant ?

C'est beau et avec de telles références, ce n'est plus la peine d'accorder « sérieux » à jeu (*serious game*) pour éviter de faire frivole. Rifkin utilise le terme « jeu profond » et le présente comme une interaction empathique avec nos semblables.

Le jeu agile n'est pas un jeu « sérieux », c'est un outil permettant d'obtenir des résultats en prenant du plaisir.



Fig. 17.5 « J'hésite entre vous donner un gage et bloquer votre bonus. »

17.5.1. Jeux pour apprendre

- **Pour acquérir la culture agile** – Ces jeux aident à la transformation agile d'une organisation en faisant percevoir la nouvelle culture, avec ses valeurs et ses principes. Parmi les très nombreux qui existent, citons *Artistes et Spécifieurs*, et le *Scrum Lego*.
- **Pour le travail en équipe** – De nombreux jeux, qui n'ont pas pour la plupart leur origine dans l'agilité, font ce qu'on appelle du *team building*, en apprenant aux membres d'une équipe à travailler ensemble.

17.5.2. Jeux pour faire le produit

Nous avons déjà évoqué nombre de ces jeux. Le plus connu est le *Planning Poker* pour estimer la taille des stories.

- **Pour la définition de produit** – L’agilité pousse au travail collaboratif : la meilleure façon de définir le produit consiste à procéder avec des ateliers ludiques. Les jeux typiquement adaptés à la définition de produit sont les *Innovation Games* [Hohmann, *Innovation Games*]. Ces jeux se pratiquent plutôt avec les utilisateurs. D’autres parties prenantes, celles côté métier, sont typiquement des participants de ces jeux.
- **Pendant les sprints** – Même si on peut s’amuser pendant les sprints, c’est surtout lors de la rétrospective qu’on utilise des jeux. Ils ont pour objectif de faciliter l’expression de tous dans la collecte des informations qu’est d’abord une rétrospective.

À côté, de nombreux « petits jeux » sont utilisés comme des techniques permettant de prendre rapidement des décisions en faisant participer tous les membres de l’équipe.

17.6. Les jeux sur le terrain

17.6.1. Ne pas lésiner sur les accessoires

Certains jeux nécessitent des accessoires. En plus du stock indispensable de Post-it®, le ScrumMaster possédera dans ses tiroirs des stocks de gommettes, même du niveau maternelle, des pastilles, de la patafix, un chronomètre (car les jeux sont aussi *timeboxés*), pour ne rester que dans les accessoires de base.

Il y a aussi la boîte à meuh utilisée pour interrompre les discussions qui sortent du cadre de la réunion, le pomodoro comme minuteur, le gizmo pour l’intégration continue, etc.

17.6.2. Prendre des photos

Souvent, ceux à qui on vante l’intérêt des Post-it® le reconnaissent bien volontiers, mais ils ajoutent qu’il faut reprendre tout ça dans une documentation, nécessaire pour les services qualité ou pour des audits.

En fait, ce n’est pas toujours vrai et quand c’est vrai, une photo peut faire l’affaire. Au-delà, l’appareil photo (ou le smartphone) est aussi un outil nécessaire pour l’agilité. Pour prendre des clichés réguliers des tableaux, mais aussi pour garder un souvenir des moments passés en équipe, en particulier lors des célébrations de fin de sprint.

17.6.3. Jouer avec les bonnes personnes

La question se pose, en particulier pour les jeux qui sont faits dans le sprint zéro : avec qui jouer ?

Une réponse simple et efficace est que les personnes qui sont là sont les bonnes personnes. C’est un principe qu’on rappelle dans les forums ouverts (*open space*). Pas la peine d’attendre qu’une personne paraissant importante soit présente, on lance l’idée du jeu et ceux qui y participent sont les bonnes personnes.

Beaucoup de jeux ont une finalité « métier » ; les parties prenantes vont naturellement y prendre part. Cependant je trouve important d’y faire participer aussi les développeurs.

Un jeu favorise les échanges, les arbitrages, la levée de certaines incompréhensions. Le format oblige les parties prenantes à aller à l'essentiel, à prioriser. Il leur permet de se libérer de leur a priori : le jeu offre un espace protégé où on a droit à l'erreur.

17.6.4. S'intéresser aux usages innovants

Le mouvement agile représente un changement radical. Il est dommage, à mon avis, de ne pas prolonger le changement dans un usage ludique de l'outil informatique. On voit encore des outils aux interfaces vieillottes, alors qu'avec les nouvelles interfaces des applications web, on peut se rapprocher de l'usage des Post-it®, avec même quelques avantages supplémentaires.

Une autre piste à suivre est l'usage qu'on retrouve dans les jeux vidéo, permettant de déplacer les stories sans clavier ni souris.

17.6.5. Créer ses jeux

Quand on a un objectif précis d'apprentissage ou de collaboration, il ne faut pas hésiter à créer un jeu pour cela, ou à adapter un jeu existant. On trouvera des idées dans [Macanufo, *Game Storming*], et en participant à des conférences où de nombreux jeux sont proposés. Il y a même une « non-conférence » dédiée aux jeux qui se tient tous les ans quelque part en France, dans laquelle on peut proposer et tester de nouveaux jeux qu'on aura mis au point.



Fig. 17.6 Le joystick sera fourni

Bien commencer

La question à se poser

Est-ce que nous avons vraiment besoin d'un outil informatique pour les stories et les tâches ?

De mauvais signes

Seul le ScrumMaster met à jour les informations du sprint.

Seul le Product Owner s'intéresse au backlog.

Les gens ouvrent leur ordinateur portable en arrivant à une réunion.

Par quoi démarrer

Préparer le tableau en équipe au cours du sprint zéro, après quelques jeux.

Une lecture pour tous

Un excellent livre regorgeant d'idées de jeux et d'ateliers [Macanufo, *Game Storming*].

À retenir

Une équipe Scrum utilise des outils simples, adaptés à son contexte.

Les notes collantes sont devenues un signe de reconnaissance de l'usage de Scrum. Pour les équipes distribuées, une application informatique peut être nécessaire pour gérer le backlog. Qu'il soit virtuel avec un outil informatique ou comme objet physique, le tableau est un outil majeur dans le management visuel推崇né par l'agilité.

Le succès d'une équipe passe par son plaisir au travail. Le jeu est l'outil essentiel qui y contribue.

Références :

☞ Agile Games France, Wiki.

<http://www.agilegamesfrance.fr>

☞ Luke Hohmann, *Innovation Games*, Addison-Wesley, 2007.

☞ Institut Agile, *Référentiel des concepts, pratiques et compétences agiles*, Web.

<http://institut-agile.fr/>

☞ James Macanufo, Sunni Brown, Dave Gray, *Gamestorming – Jouer pour innover*, VF, 2014.

<http://www.diateino.com/fr/90-gamestorming.html>

☞ Jeremy Rifkin, *La troisième révolution industrielle*, Les liens qui libèrent, 2012.

Notes

- [1] La cohabitation de Post-it® avec un outil est tout à fait possible. Voir ce billet : <http://www.aubryconseil.com/post/Scrum-avec-un-outil>.
- [2] <http://www.bouzin-agile.fr/?post/2010/10/15/Evitez-d'avoir-le-post-it-qui-rebique>
- [3] Voir le billet « les tâches en couleurs » : <http://www.aubryconseil.com/post/Les-taches-en-couleurs>
- [4] Indicateur visuel du moral de l'équipe : <http://referentiel.institut-agile.fr/nikoniko.html>. On peut aussi le pratiquer en ligne : <http://teammood.com>.
- [5] <http://www.microsoft.com/en-us/download/details.aspx?id=4401>

Améliorer la visibilité avec des indicateurs

L'approche empirique de Scrum, basée sur l'inspection et l'adaptation, nécessite d'avoir de la visibilité sur ce qui est fait par l'équipe pour réaliser le produit. Le management visuel recommandé à travers les chapitres précédents permet déjà de se rendre compte de la situation courante, à tout moment. Les tableaux montrant les tâches, les stories et les features présentent des informations très parlantes, pour qui a un peu l'habitude.

Du coup, l'importance accordée d'habitude aux indicateurs (les fameux KPI, *Key Performance Indicators*) est moins prégnante. Des indicateurs sont cependant utiles pour améliorer la visibilité, en apportant des informations que les tableaux instantanés ne montrent pas : des tendances.

Un indicateur permet de renforcer, ou pas, la confiance dans la tenue d'un objectif de sprint ou de release.

L'indicateur historique de Scrum est le *burndown chart*, qui montre l'évolution du reste à faire. Souvent mal utilisé, il présente en outre des limitations dans certains contextes pour lesquels d'autres indicateurs, que nous allons présenter dans ce chapitre, sont plus pertinents.

Pour produire des indicateurs, il faut collecter des mesures brutes. Avec Scrum, les mesures les plus importantes portent sur des résultats concrets, collectés au rythme des boucles d'inspection et d'adaptation : tous les jours, tous les sprints, toutes les releases. Elles permettent de construire des indicateurs utiles pour prendre des décisions et pour analyser des effets au cours des mêlées, des revues et des rétrospectives.

Tous les indicateurs présentés ci-après ne sont pas à collecter dans tous les projets, c'est à l'équipe de définir ceux qui doivent l'être en fonction de ses objectifs et de ses possibilités.

Certaines mesures portent sur des grandeurs qui ne sont pas connues au moment où on en a besoin et qu'il faut donc estimer. C'est le cas des points de story. Nous reviendrons, dans une deuxième partie, sur ce sujet, qui fait l'objet de nombreuses discussions dans la communauté agile : les estimations.

18.1. Indicateurs pour le suivi du sprint

Pendant un sprint, la collecte de tous les jours peut porter sur :

- le nombre de tâches finies ;
- les vérifications de la définition de fini faites avec succès ;
- l'humeur de l'équipe.

La mesure quotidienne des stories finies ne paraît pas utile, car cela se devrait se voir aisément sur le tableau du sprint.

18.1.1. Burndown de sprint

Dans sa forme historique, le *burndown chart* était réalisé avec les heures restant à faire. Cependant, l'estimation des tâches en heures constitue du gaspillage ; je conseille simplement de compter les tâches pour produire un indicateur.

J'ai vu tellement de burndown charts de sprint, censés descendre, qui remontaient, pour des raisons que les personnes concernées avaient du mal à expliquer (même si je devinais pourquoi), que je ne conseille pas cet indicateur. Il ne répond pas à au moins une règle des « 3A » (<http://jem9.com/3-metrics/>) : actionnable, accessible, auditabile.

18.1.2. Burnup de sprint

Un indicateur intéressant, obtenu avec la mesure sur les tâches, est le *burnup* de sprint en tâches.

J'ai remarqué que la plupart des gens préfèrent les courbes qui montent à celles qui descendent. C'est moins perturbant de représenter l'avancement plutôt que le reste à faire.

Usage	Uniquement destiné à l'équipe dans le suivi de son sprint. N'est pas conservé au-delà du sprint.
Quand l'utiliser ?	Quand l'équipe débute. On arrête quand l'équipe arrive régulièrement au succès des sprints.
Collecte	Ce qui est fini, mesuré tous les jours.
Unité	En nombre de tâches.
Tendance souhaitée	Montée régulière.
Variante	Avec le burnup simple, on ne voit pas les tâches ajoutées, comme les urgences non prévues. C'est pourquoi on peut ajouter une deuxième courbe, avec le total. L'objectif est que les deux courbes se rejoignent.

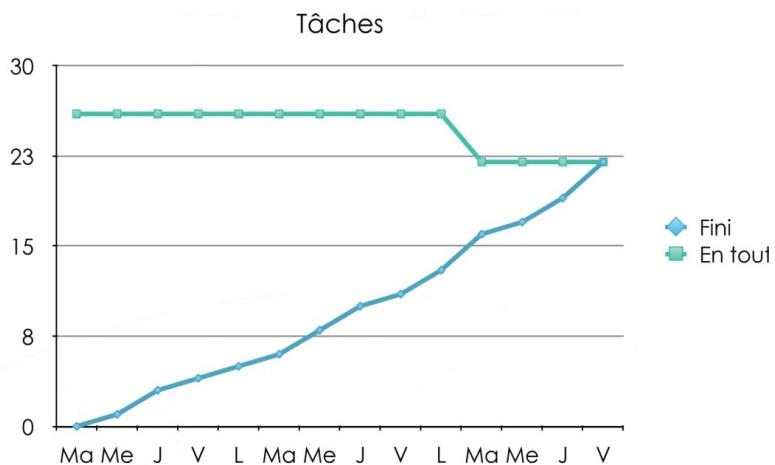


Fig. 18.1 Un burnup de sprint en tâches.

Une autre variante, pour les équipes aguerries, est de mesurer le nombre de vérifications passées avec succès sur les stories, en relation avec la définition de fini. Cela permet de mieux se rendre compte du degré de finition des stories.

18.1.3. Suivi de l'humeur

Un nouveau type de mesure encouragé par les méthodes agiles concerne la satisfaction des gens.

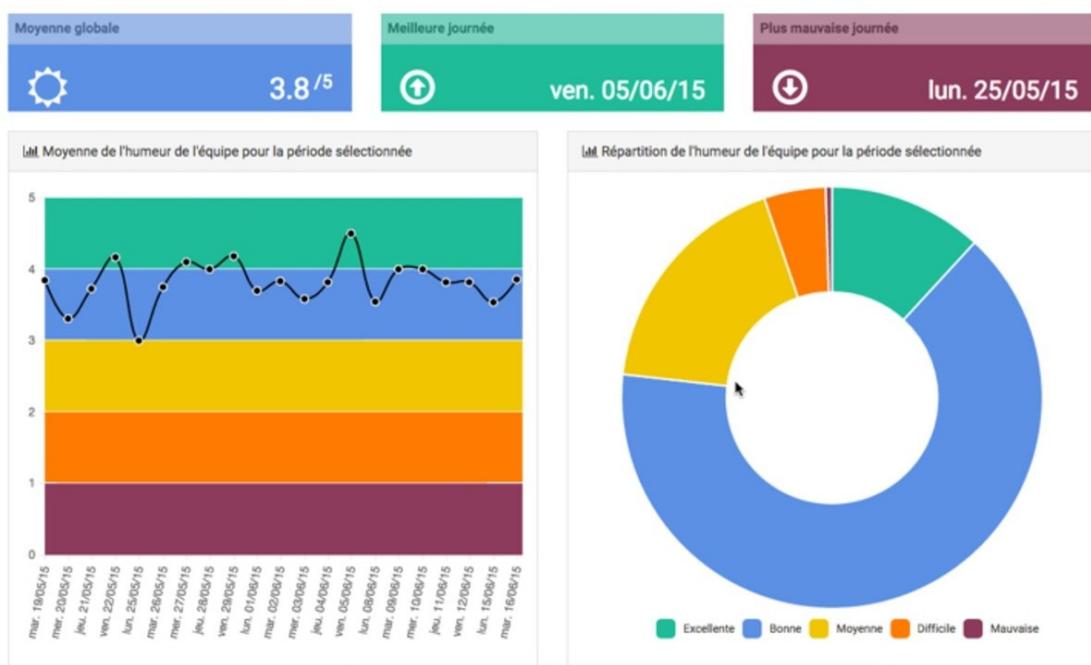


Fig. 18.2 Indicateur d'humeur produit avec Teammood (teammood.com)

Usage	Uniquement destiné à l'équipe dans le suivi de son sprint. Peut être analysé lors de la rétrospective.
Quand l'utiliser ?	Quand l'équipe se forme. On arrête quand l'équipe considère que ce n'est plus utile.

Collecte	L'humeur de chaque membre de l'équipe, collectée tous les soirs, en quittant le travail.
Unité	À définir par l'équipe ; par exemple choix entre cinq valeurs : excellente journée, bonne journée, journée moyenne, journée difficile, mauvaise journée.
Tendance souhaitée	Bonne humeur.
Risques	Personnalisation excessive. C'est pourquoi on peut le faire de façon anonyme.

18.2. Indicateurs relatifs à l'équipe

18.2.1. Vélocité de sprint

La vélocité est la partie du backlog réalisée en un sprint. Elle sert à prévoir la capacité pour les prochains sprints.

Usage	Pour estimer la capacité de l'équipe en vue de la planification de la release.
Quand l'utiliser ?	Dès que possible et tout au long du développement.
Collecte	Vélocité mesurée à chaque fin de sprint.
Unité	En nombre de stories ou en points.
Tendance souhaitée	Stabilité, afin d'avoir confiance dans les prévisions.
Risque	Tendance pendant le sprint, tendance à vouloir une croissance continue de la vélocité, ce qui peut nuire à la qualité et provoquer de la dette technique.

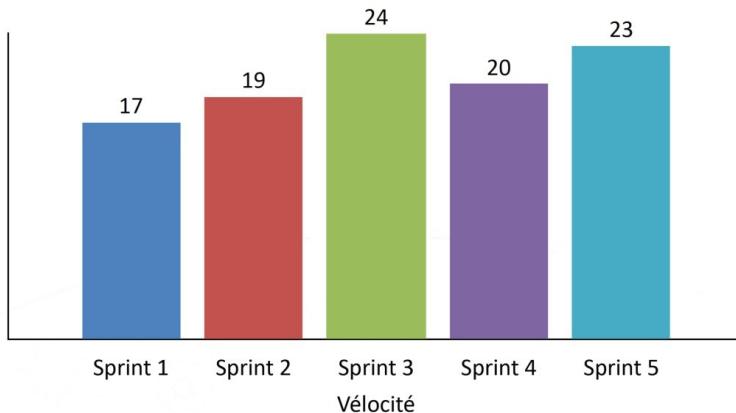


Fig. 18.3 Indicateur de vélocité des sprints

18.2.2. Suivi des obstacles

Un obstacle est un fait concret qui ralentit ou freine l'équipe.

Usage	Pour évaluer l'aptitude de l'équipe à éliminer les obstacles. Vient en complément au tableau des obstacles.
Quand l'utiliser ?	Quand il y a beaucoup d'obstacles et qu'ils traînent. On arrête quand l'équipe arrive à les éliminer régulièrement.
Collecte	Le nombre d'obstacles non éliminés, collecté en fin de sprint.
Tendance souhaitée	Décroissance, puis stabilité.
Variante	La distinction entre les obstacles internes et externes peut être utile pour la rétrospective.

18.3. Indicateurs pour le suivi de la release

18.3.1. Burndown de release

Le plan de release présente le travail qui reste à faire pour la release. Un burndown montre la tendance.

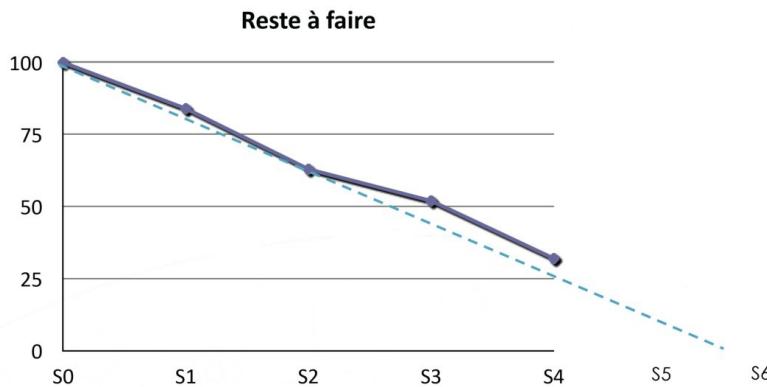


Fig. 18.4 Un burndown chart de release

Usage	Présentation lors de la revue.
Quand l'utiliser ?	Dès que possible et tout au long du développement. Il est présenté lors de la revue de sprint.
Collecte	À chaque sprint : la valeur de l'ordonnée correspond à la partie qui reste à faire d'ici la fin de la release.
Unité	En nombre de stories ou en points. Ou en nombre de features.
Tendance souhaitée	Décroissance linéaire.
Limites	On ne voit pas l'influence des changements de périmètre, ce qui rend le graphe difficile à comprendre ; en cas de variation de périmètre importante, la courbe peut remonter.

Si on « déscope » régulièrement, le burndown suffit. En revanche, si on n'a pas cette habitude, un burnup à deux courbes sera préférable pour montrer que des stories ont été ajoutées au bac d'affinage.

18.3.2. Burnup de release à deux courbes

Le burnup possède deux courbes : une qui montre ce qui est fini et l'autre qui représente tout le travail contenu dans le backlog, y compris ce qui est fini. La première ne descend jamais, il n'y a pas de vitesse négative ! La seconde peut monter mais aussi descendre : si on supprime des stories qui avaient déjà été estimées, ou si on réestime à la baisse.

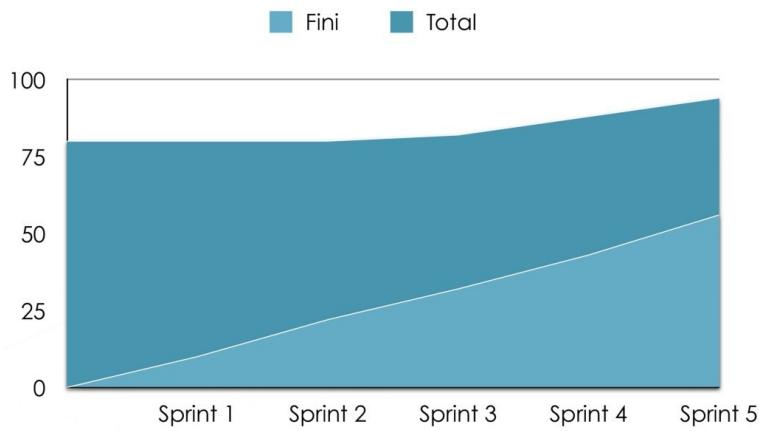


Fig. 18.5 Un burnup chart de release

On voit que la taille de la release augmente à partir du sprint 3, suite au feedback. Ce serait sûrement le moment de déplacer des stories dans le bac à glace.

Usage	Présentation lors de la revue.
Quand l'utiliser ?	Dès que possible et tout au long du développement.
Collecte	À chaque sprint : ce qui est fini depuis le début de la release et le total du périmètre de la release.
Unité	En nombre de stories ou en points.
Tendance souhaitée	Croissance linéaire de la première courbe, qui rejoint la deuxième avant la fin de la release.
Limites	On ne voit pas les incertitudes. C'est pourquoi on ira voir le plan de release, qui les montre.

18.4. Pas d'indicateur de productivité

La mesure de la vélocité permet de produire des indicateurs. Peut-on s'en servir pour évaluer la productivité ?

18.4.1. La vélocité n'est pas de la productivité

La vélocité repose sur des estimations

L'estimation est, depuis toujours, une activité extrêmement délicate dans le développement de logiciel. Il n'y a pas de modèle universel, le meilleur outil pour estimer se trouve dans l'historique des données collectées (les mesures).

L'estimation avec les méthodes agiles a apporté des réponses nouvelles mais suscite toujours beaucoup de discussions. C'est un sujet qui, finalement, n'est pas simple.

Avec Scrum et les méthodes agiles, l'estimation reste un art difficile, mais la façon de l'aborder est différente :

L'estimation est collective, elle est faite par ceux qui réalisent.

C'est mieux. Malgré cela, les points de story comportent une part importante d'incertitude. La vélocité moyenne, utilisée pour planifier la release, est plus fiable, mais la vélocité d'un sprint reste une mesure incertaine.

La vélocité mesure du travail

Que mesure vraiment la vélocité ? Voyons d'abord l'unité utilisée.

Il ne s'agit pas d'hommes-jours

Tous les sprints ayant la même durée avec une équipe stable, prendre cette unité aboutirait à une vélocité constante par définition. La subtilité de la vélocité est qu'elle mesure des choses (stories) finies et que sa variation est une indication utile.

Il s'agit de points de story

Si on pratique l'estimation au moment de l'approvisionnement (comme présenté dans le chapitre 7 *Affiner le backlog*), il y a de bonnes chances qu'on soit sur les points de story, basés sur la **difficulté intrinsèque**.

Quoi qu'il en soit, cela est en rapport avec du travail, des choses finies. Ce n'est pas de la productivité.

Contrairement à des idées répandues, vélocité et productivité sont deux mesures différentes. La définition classique de la productivité, c'est le quotient du résultat sur le temps passé à produire. La notion est surtout utilisée en économie pour montrer que l'utilisation de machines permet de réduire le temps de production.

Comme la définition de la productivité parle de résultat, c'est la valeur ajoutée qu'il faudrait utiliser. La mesure de la valeur apportée par chaque sprint, faite de cette façon, se rapprocherait plus de la productivité au sens utilisé en économie. Ce n'est donc pas parce que la vélocité augmente que la productivité va augmenter.

Valeur et taille

Dit autrement, ce n'est pas parce qu'une story est grosse qu'elle apporte beaucoup de valeur.

Sur un grand échantillon de stories, la taille et la valeur sont statistiquement corrélées : en moyenne, plus la taille est grande plus la valeur est importante (figure 18.6, XYZ).

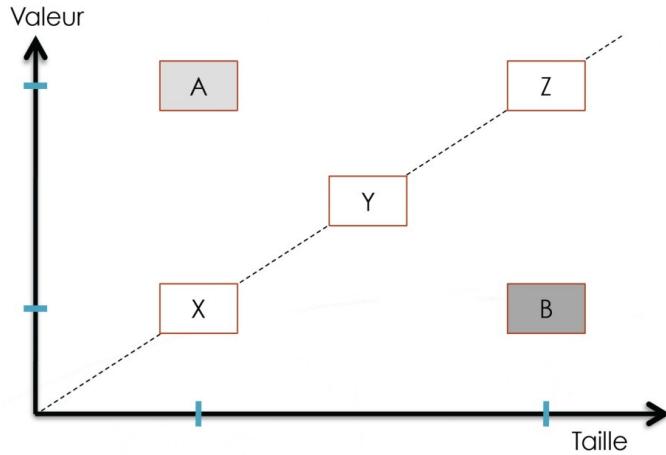


Fig. 18.6 Corrélation de valeur et taille.

Mais ce n'est évidemment pas vrai pour tout (figure 18.6, A possède plus de valeur que X pour la même taille) : on connaît tous l'exemple de fonctions faciles à développer qui peuvent être très utiles (ou inversement des « usines à gaz » longues à réaliser et qui finalement ne servent à rien).

Puisque la vélocité n'est pas le reflet de la productivité, essayons avec la valeur métier.

18.4.2. La valeur métier est subjective

C'est un but proclamé de Scrum et des méthodes agiles : chercher à maximiser la valeur métier. Plus la valeur d'un élément est importante, plus sa priorité est élevée et, comme la priorité définit l'ordre dans lequel les éléments du backlog sont réalisés, les éléments avec le plus de valeur sont développés en premier.

La valeur repose aussi sur des estimations

Mais ce n'est pas facile d'estimer la valeur ajoutée par une story...

Il faut déjà définir ce qu'on entend par la valeur pour le métier (*business value*) : le retour sur investissement, la valeur actuelle nette ? Ensuite, même avec un gros travail d'étude, estimer la valeur financière que va rapporter une story est une gageure.

Comme l'estimation de la taille, celle de la valeur peut se faire en points, sans unité et de façon relative. Le Product Owner fait, implicitement, un tri selon la valeur des stories quand il ordonne par priorité (c'est ce qu'on appelle l'utilité ordinaire).

Pour aller plus loin que l'ordre et obtenir une mesure, il faudrait évaluer la valeur de chaque élément (faire de l'utilité cardinale).

La valeur a plus de sens pour une feature

Une user story apporte de la valeur métier. À la différence des user stories, les stories techniques et la dette technique n'ont pas une utilité perceptible par des utilisateurs. Cependant, comme des user stories en dépendent, il est possible de leur allouer une valeur indirecte. Les bugs, eux, retirent de la valeur à la story sur laquelle ils portent.

Cependant, on s'aperçoit très vite qu'au niveau de la story, il ne s'agit pas vraiment de valeur métier. Une story est trop petite pour être déployée individuellement. Il est plus raisonnable de définir la valeur sur des features.

Problèmes avec la mesure de la valeur

La valeur est une notion très subjective. Certes des techniques et des ateliers permettent de classer des éléments par valeur, mais se servir d'un chiffre même relatif pour des indicateurs est sujet à caution.

C'est pourquoi, même si l'estimation relative de la valeur est possible, je ne montrerai pas d'indicateur basé sur la valeur, qui serait trop subjectif dans son interprétation.

La théorie présente une courbe en S (citée par Alistair Cockburn)^[1], mais cela reste de la théorie, juste pour expliquer que la valeur métier monte bien, une fois qu'on a réduit les risques (valeur de connaissance), et qu'elle atteint un palier dû à la valeur marginale si on reste à périmètre constant.

De la valeur à l'utilité ou l'impact

En outre, la valeur est une notion mal comprise : on s'aperçoit que beaucoup la confondent avec le coût.

Pour éviter les confusions, il serait envisageable de changer de vocabulaire et de parler d'**utilité**. C'est une notion utilisée en économie^[2] : l'utilité est une mesure du bien-être ou de la satisfaction obtenue par la consommation, ou du moins l'acquisition, d'un bien ou d'un service. Cela aurait aussi l'avantage d'être plus général : si tous les produits ne visent pas à apporter de la valeur financière, ils ont tous vocation à être utiles. C'est le cas des logiciels open source.

Ce que nous enseigne l'*impact mapping* (chapitre 14 *Découvrir le produit*), c'est qu'il serait encore mieux de se préoccuper des impacts sur les acteurs et que, plutôt que de se baser sur des estimations discutables, il vaudrait mieux mesurer les impacts réels.

En résumé

Il n'y a pas d'indicateur sérieux de productivité avec Scrum, qu'il soit basé sur la vélocité ou la valeur métier.

La vélocité sert principalement à mieux prévoir l'avenir, et la valeur métier à mieux prioriser les éléments.

18.5. Pas d'indicateur du niveau d'agilité

Des organisations se demandent parfois si elles sont devenues agiles ou, parfois pour se comparer, quel niveau d'agilité elles ont atteint.

On pourrait imaginer aussi de comparer le niveau d'agilité de plusieurs équipes d'une organisation. Pour cela, certains proposent d'évaluer les pratiques mises en place par une

équipe. Les spécialistes processus sont prêts à accourir pour mesurer la mise en œuvre des « bonnes » pratiques, produire des indicateurs et en déduire un niveau d’agilité.

Cette approche processus n’est pas adaptée à Scrum. Comme nous l’avons vu dans le chapitre 13 *Contextualiser Scrum*, chaque situation est différente. Nous ne proposerons pas d’indicateur de suivi des pratiques. Il serait plus dans l’esprit de l’agilité d’évaluer la satisfaction des gens et les résultats obtenus.

La satisfaction des développeurs de l’équipe, nous l’avons évoquée avec l’indicateur montrant l’humeur quotidienne. Cela pourrait être élargi aux parties prenantes sur le rythme du sprint.

Les résultats, c’était l’objet de notre discussion précédente sur la productivité. Nous ne nous risquerons donc pas à conseiller des indicateurs processus relatifs au niveau d’agilité.

Nous conseillons cependant de penser son chemin au travers de l’agilité. Le sujet sera approfondi dans le chapitre 22 *Transformer les organisations*.

18.6. Les indicateurs sur le terrain

18.6.1. Attention à ne pas se tromper dans la cible

Certains indicateurs sont destinés à l’équipe et n’intéressent pas les parties prenantes, comme le burndown chart de sprint.

Il ne suffit pas de produire des graphiques, il convient de sélectionner les plus pertinents pour les personnes auxquelles ils sont destinés. Les plus simples sont les meilleurs.

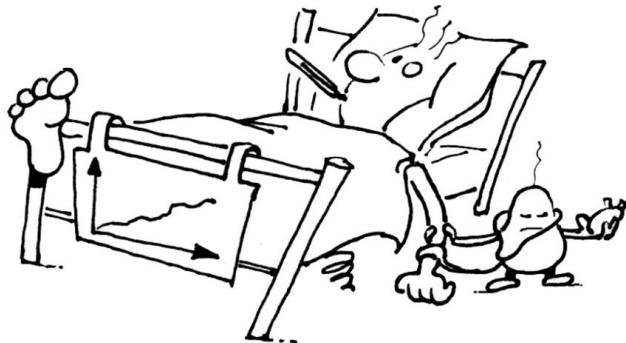


Fig. 18.7 Un indicateur simple

Des indicateurs simples et en nombre limité, cela facilite la transparence et évite au ScrumMaster de passer du temps à les construire, puis à les expliquer, en particulier à ceux qui ont l’habitude de la gestion de projet traditionnelle.

18.6.2. On ne mesure pas le temps consommé

La pratique habituelle de gestion de projet consiste à estimer la durée de la tâche avant de la commencer, de relever les heures passées effectivement et d’analyser les écarts constatés.

Avec Scrum, on ne se préoccupe pas des heures consommées ni sur la tâche ni sur la story. Ce qui compte, c'est finir la tâche et surtout la story.

J'entends parfois « *à quoi sert de faire des estimations si on ne mesure pas le temps effectivement passé ?* ». Les estimations servent d'abord à planifier. On peut très bien avoir des estimations sans mesure du temps réellement passé. C'est exactement ce qu'on fait avec Scrum. Ce qui nous intéresse, c'est le reste à faire, pas le relevé des heures. C'est avec l'évolution du reste à faire que des décisions de planification peuvent être prises, par exemple ajuster l'objectif d'un sprint en ajoutant ou supprimant une story.

La mesure individuelle du temps passé présente de nombreux inconvénients : elle prend du temps, elle n'est pas fiable, elle pousse à considérer que l'objectif d'un projet est de tenir une estimation plutôt que de produire quelque chose, elle conduit à la spécialisation au détriment de la pluridisciplinarité de l'équipe. Elle nuit à l'esprit collectif.

Enfin, cela ne sert pas à grand-chose. En effet, si on observe un décalage entre une estimation et les heures effectivement passées, on ne peut pas dire si c'est un problème d'estimation ou de compétence ou de définition du travail.

Quand une équipe applique Scrum pour la première fois, on constate que certains de ses membres ont beaucoup de réticences à donner une estimation. Une des raisons est la peur d'être jugé sur la qualité de leur estimation et sur leur productivité individuelle. Cette tendance – naturelle – à considérer une estimation comme un engagement diminue quand le relevé des heures consommées n'est pas mis en exergue et qu'il est substitué par la volonté de finir. Cela est aussi atténué par la pratique de l'estimation collective.

18.6.3. Estimer la taille plutôt que l'effort

L'estimation est en points, mais basée sur quoi ? Par rapprochement avec les estimations habituelles, certains parlent de points d'effort. D'autres évoquent les points de complexité.

Il vaut mieux éviter les points d'effort. L'effort est très mal connu au moment où une story entre dans le bac d'affinage. Compter en points d'effort obligerait à revoir les points souvent et en particulier au moment de la planification de sprint, voire pendant le sprint.

Pour avoir participé à de très nombreuses sessions de planning poker, je me suis rendu compte que ce qui était estimé assez naturellement par les participants, c'était ce qu'on peut appeler la difficulté intrinsèque relative, pas l'effort.

Certaines équipes utilisent les points d'effort pour calibrer le sprint : elles estiment les stories au moment de la planification de sprint pour définir combien en caler dans le sprint. Je déconseille cette pratique pour le sprint, préférant l'engagement sur un objectif (voir le chapitre 9 *Planifier le sprint*).

Le temps que va prendre une story pour être réalisée (l'effort) dépend de notions comme la dette technique et la constitution de l'essaim, qui ne sont pas connues au moment de l'estimation, laquelle peut avoir lieu longtemps avant la réalisation. C'est pourquoi les estimations portent uniquement sur la perception de la difficulté intrinsèque de la story,

relativement aux autres. Cela évite de mettre à jour l'estimation d'une story si on se rend compte que la dette technique a augmenté.

Dans une édition précédente, je parlais d'effort et non pas de taille. J'ai changé d'avis suite à mes expériences sur le terrain, confortées par la lecture du livre *Exploring Scrum* [Rawsthorne, *Exploring*]. La notion de difficulté intrinsèque relative (qui vient de Ron Jeffries) y est reprise de façon convaincante.

L'estimation en taille, basée sur la difficulté intrinsèque relative, inclut le volume d'informations à traiter dans une story, dont la complexité peut être simple mais avec de nombreuses occurrences. Il ne s'agit pas de difficulté technique uniquement, cela concerne toutes les activités pour finir la story.

Bien commencer

La question à se poser À qui sont destinés les indicateurs ? Qu'en attendent-ils ?

Les indicateurs sont produits par l'outil informatique et sont incompréhensibles.

De mauvais signes Les indicateurs de sprint sont présentés aux parties prenantes lors de la revue.

Les mesures ne sont pas faites au bon moment et faussent l'interprétation.

Par quoi démarrer Commencer par un burnup de release.

Une lecture pour tous *Petit cours d'autodéfense intellectuelle*, pour bien compter et ne pas s'en laisser conter avec les indicateurs [Ballargeon, *Autodéfense*].

À retenir

Les mesures agiles sont différentes. Elles diffèrent dans leur nature avec l'importance donnée aux résultats visibles (vitesse mesurée avec les stories finies).

Les indicateurs, élaborés à partir des mesures, sont utiles pour le suivi des plans ; ils servent aussi à l'équipe pour qu'elle évalue ses progrès et améliore ses pratiques.

Référence :

☞ Normand Ballargeon, *Petit cours d'autodéfense intellectuelle*, 2006, éditions Lux.

☞ Dan Rawsthorne & Doug Shimp, *Exploring Scrum*, 2012.

<http://agilelib.net/?100002>

Notes

- [1] <http://alistair.cockburn.us/Develop+for+business+value+once+risks+are+down.png>
- [2] <http://fr.wikipedia.org/wiki/Utilit%C3%A9>.

Ajouter les pratiques de développement XP

Scrum s'est largement diffusé pour les développements de logiciel, avec d'indéniables succès. Cependant, et malgré les alertes régulières sur les risques « *Scrum sans pratiques d'ingénierie, c'est risqué, attention à la dette technique !* », on constate que certaines équipes ont toujours des difficultés à obtenir des produits de qualité.

Scrum ne présente pas explicitement de pratiques de développement, mais leur usage est induit par la définition de fini : pour respecter ce que signifie fini, une équipe est poussée à les appliquer.

Ce serait une erreur de considérer que ces pratiques sont optionnelles et que la qualité du produit n'est pas un objectif de Scrum. Le code doit être de la meilleure qualité possible. Scrum n'est pas compatible avec le développement « vite fait, mal fait ».

L'objectif de ce chapitre est de présenter comment Scrum doit être complété avec des pratiques de développement logiciel. Nous aborderons rapidement les plus courantes, et nous montrerons surtout comme elles s'intègrent au cadre Scrum.

19.1. Pratiques autour du code

Nous allons aborder uniquement des pratiques issues *d'Extreme Programming* (XP). Cependant, des pratiques de développement plus anciennes, comme la gestion de versions, le suivi de règles de codage, la revue de code, etc. sont elles aussi nécessaires pour garantir la qualité d'un développement de logiciel.

19.1.1. Intégration continue

L'intégration continue est une pratique de développement logiciel qui conduit les membres d'une équipe à intégrer leur travail fréquemment ; habituellement chaque personne le fait au moins une fois par jour, ce qui conduit à avoir plusieurs intégrations quotidiennes.

Une pratique indispensable

L'intégration régulière du code de chaque développeur est une pratique essentielle pour le développement itératif.

Comme le dit Martin Fowler, qui a contribué à l'essor de la pratique^[1] : « *Je crois que toutes les équipes devraient pratiquer l'intégration continue. Les outils sont gratuits. Le seul prix à payer c'est d'apprendre* ».

Une intégration produit un *build*, résultat de la compilation et de l'assemblage des composants, utilisé pour passer des tests et permettre le feedback du Product Owner. Du

point de vue Scrum, cela correspond à un micro-incrémentation qui aboutira en fin de sprint à l'incrément de produit.

À chaque *commit*^[2] d'un développeur, un outil placé sur le serveur d'intégration (par exemple Jenkins ou Travis) lance une série d'actions. L'enchaînement est généralement le suivant :

- Compiler et vérifier.
- Lancer les tests unitaires.
- Lancer des tests d'intégration et les tests d'acceptation.
- Lancer l'outil qui vérifie le style d'écriture (par exemple *Linter*).
- Lancer l'outil qui vérifie la qualité du code (par exemple *Sonar*).
- Produire un rapport, avec les erreurs éventuelles.
- Empaqueter une version de logiciel prête à l'emploi.

L'intégration continue garantit que le code récemment ajouté fonctionne bien avec l'incrément précédent.

Si l'intégration ne passe pas, l'équipe s'arrête pour identifier rapidement le problème à l'origine de l'erreur et relancer la fabrication : il ne faut pas laisser une intégration « cassée », pour éviter de propager des erreurs.

Pour qu'une intégration en échec soit réparée rapidement, on peut utiliser un accessoire^[3].

Quand la mettre en place ?

Il est préférable que l'intégration continue soit maîtrisée avant le premier sprint. Si l'équipe n'en dispose pas et décide de la mettre en place pendant une release, l'installation du serveur et du logiciel constitue une story technique qui va dans le backlog de produit. Ce sera à l'équipe de négocier sa priorité avec le Product Owner, en lui expliquant tout l'intérêt qu'il va en tirer, notamment de donner son feedback plus rapidement.

Bénéfices

L'intégration continue permet d'avoir à tout moment un logiciel qui fonctionne, ce qui motive l'équipe et aussi les utilisateurs. Elle augmente la transparence en évitant que des travaux d'une personne restent ignorés du reste de l'équipe pendant un certain temps.

Déploiement continu

Une nouvelle pratique, qui prolonge l'intégration continue, est le déploiement continu. Elle consiste à déployer le résultat de l'intégration dans un environnement accessible par des parties prenantes ou des utilisateurs finaux.

Cela permet d'obtenir du feedback encore plus rapidement et de passer les tests dans un environnement plus représentatif.

19.1.2. Développement piloté par les tests

Un développeur qui écrit du code a pour responsabilité de tester son code morceau par morceau : c'est ce qu'on appelle le test unitaire.

La pratique du développement piloté par les tests [Beck, TDD] va plus loin : il s'agit d'écrire les tests avant d'écrire le code et de profiter de la présence de tests automatiques pour améliorer le code en toute sécurité.

Test en premier

Le programmeur écrit d'abord les tests unitaires d'un composant : cela lui permet de réfléchir au comportement attendu de ce composant. En principe, il n'écrit qu'un test à la fois. Une fois ce test écrit, il écrit le code pour que le test soit passé avec succès ; avoir écrit le test avant permet de rester simple au niveau du code. Le programmeur continue ainsi en ajoutant de nouveaux tests, puis le code minimal pour qu'ils passent. *De facto*, cela produit un code testable.

Cette pratique est en fait plus une méthode de conception que de codage, dans la mesure où la réflexion sur le test guide le développeur vers une solution. Elle favorise la conception émergente et la prise de connaissance du code, c'est une forme de documentation implicite.

Le pilotage par les tests inclut la pratique de remaniement de code (*refactoring*).

Remaniement du code

Le remaniement de code consiste à améliorer le code sans changer son comportement. L'objectif est d'améliorer sa qualité en simplifiant et optimisant la solution actuelle : on élimine la duplication de code et on le rend plus simple et plus facile à lire. Et donc à maintenir.

À l'issue de chaque modification, il convient de lancer les tests à nouveau pour s'assurer que le comportement reste celui attendu. Le remaniement de code peut se pratiquer sans élaborer les tests en premier, mais c'est l'intégration des deux qui constitue le pilotage par les tests.

TDD = Test écrit en premier + remaniement du code

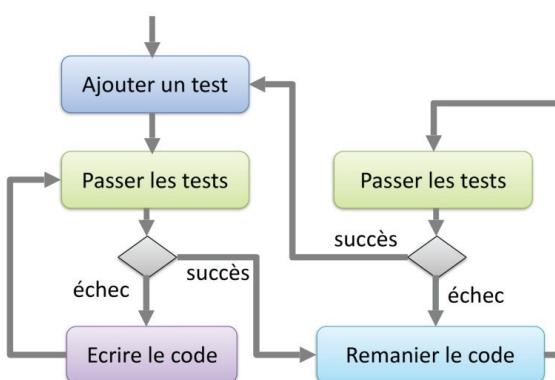


Fig. 19.1 La pratique du pilotage par les tests

Quand commencer ?

Éviter la dette technique

Après s'être formé, le mieux est de mettre en place les conditions pour faire du pilotage par les tests dès le début d'un développement, avant le premier sprint d'une release. Le niveau de pratique souhaité est renseigné dans la définition de fini : par exemple, si l'équipe considère qu'il est nécessaire d'avoir des tests unitaires pour certaines parties du code, c'est l'endroit où elle le rend explicite.

Dans le plan de sprint, des tâches de test unitaire peuvent être identifiées à part ou incluses dans celles de codage, selon l'expérience de l'équipe : si elle est novice, il est préférable de les rendre explicites.

Rembourser la dette technique

Si l'équipe a repris un logiciel existant, il existe probablement un passif, c'est-à-dire que des parties du logiciel n'ont pas de tests unitaires et ont besoin d'être remaniées.

La question qui se pose est la résorption de ce passif, appelé aussi **dette technique**.

Avant de se consacrer au remboursement de la dette, il convient de définir les composants qui doivent être remaniés et de les ordonner par priorité.

Le travail à faire pour remanier et écrire des tests sur des parties existantes prend du temps, car généralement le code est difficilement testable. C'est pourquoi les travaux doivent être identifiés comme des stories et ont leur place dans le backlog. Le Product Owner doit être impliqué pour que ces stories, qui visent à améliorer la qualité du produit, soient prises en compte au bon moment, généralement au plus tôt.

19.1.3. Programmation en binôme

La programmation en binôme est une pratique qui consiste à se mettre à deux développeurs devant un seul poste de travail, de façon à ce que la collaboration améliore la qualité du code.

Un développeur tient le clavier et programme pendant que l'autre propose des initiatives et réagit en observant l'écran. La collaboration naît de la différence des points de vue : le premier est orienté vers les détails du code tandis que le second a le recul sur la structure de l'ensemble du programme.



Fig. 19.2 Un récital de programmation en binôme à un seul clavier.

Quand en faire ?

C'est à l'équipe de décider de quelle façon elle met en œuvre la pratique de binômage. Il n'est pas possible d'en faire toute la journée, car c'est très intense. Il est nécessaire de permutez régulièrement les rôles dans un binôme et entre les personnes pour les binômes de l'équipe. Cela s'organise lors des mêlées quotidiennes.

C'est particulièrement conseillé sur les parties ardues du logiciel. Cela peut aussi se pratiquer pour le transfert de compétences dans l'équipe.

Bénéfices

Le travail en binôme contribue à améliorer la qualité du produit : la dette technique sera moindre sur les parties réalisées en duo.

C'est la quintessence de la collaboration.

Binômes en dehors de la programmation

Cette pratique peut être étendue au-delà des développeurs : la constitution de binômes développeur-testeur s'avère également efficace.

C'est pourquoi le binômage (travail en binôme) est plus une pratique de collaboration qu'une pratique de programmation. En fait, c'est une variante de l'essaimage.

19.2. Pratiques de conception

Autrefois, on distinguait la conception préliminaire de la conception détaillée. Maintenant, le terme « architecture du logiciel » est largement employé. L'architecture est relative à l'organisation des composants et à leurs interactions. Pour faire simple, l'architecture est

globale et permet de guider des choix de conception faits sur un composant ou pour développer une story.

19.2.1. Architecture évolutive

En caricaturant les points de vue, on dirait qu'avec une méthode traditionnelle, on est censé élaborer toute l'architecture au début et qu'avec une méthode agile, l'architecture commence avec la première itération et se poursuit régulièrement.

Dans un cadre Scrum, s'il n'est pas recommandé d'être dans le premier cas en figeant très tôt l'architecture, il n'est pas interdit de faire de l'architecture pendant le sprint zéro. Le temps à y consacrer dépend de chaque contexte. En tout cas, l'horizon de la réflexion ne doit pas dépasser le cadre de la release courante, pour éviter de prendre des décisions prématurées.

Un des objectifs du sprint zéro est de prouver que l'architecture est maîtrisée. Pour cela, on peut développer une story significative du point de vue de l'architecture, c'est-à-dire passant par les composants essentiels pour réduire le risque.

Quelle que soit la part d'architecture faite avant le premier sprint, il faut partir sur l'idée qu'il y aura encore des travaux à mener pendant les sprints : l'architecture est évolutive.

Les gros travaux constituent des stories techniques et vont dans le backlog de produit. Des travaux d'architecture peuvent demander l'assistance ponctuelle d'un expert, il convient d'anticiper leur planification pour s'assurer de sa disponibilité. Comme pour toutes les stories qui n'apportent pas de valeur visible, une négociation avec le Product Owner est nécessaire pour le convaincre de l'importance de ces travaux pour l'ordonnancement par priorité.

Le document d'architecture, s'il existe, est mis à jour à chaque sprint et fait l'objet d'une entrée dans la définition de fini pour le sprint. Mais le point important est que l'architecture soit bonne, et ce n'est pas un document qui va permettre de le savoir.

C'est en écrivant du code qu'on prouve l'architecture.

19.2.2. Conception émergente

La pratique de conception émergente se concrétise par des travaux de conception réalisés régulièrement, à chaque sprint. Nous avons évoqué deux moments où de la conception était effectuée pendant les sprints :

- Pendant la planification de sprint, plus précisément lors de l'essaimage, la conception d'une story est préparée. Elle fait l'objet d'un travail collectif, qui peut être approfondi pendant le sprint. Le résultat peut être collecté dans un diagramme de séquence montrant les interactions entre les composants collaborant à la réalisation de la story, ou selon tout autre moyen qui conviendra à l'essaim.

- Pour développer une story, l'écriture des tests en premier (pour le pilotage par les tests) participe à la conception ; cette activité est menée par un développeur ou en binôme.

Il peut aussi être nécessaire de mener des travaux d'étude ou d'exploration technique pendant un sprint. C'est ce qu'on appelle un *spike*. Le besoin est identifié lors de l'affinage. Le *spike* est alors ajouté au backlog et priorisé.

À la fin du sprint incluant le *spike*, on devrait avoir défini une solution, et être capable de rendre prête la story objet de l'étude.

19.3. Maintenance

19.3.1. Il n'y a pas de phase de maintenance

Dans de grandes organisations, on sépare traditionnellement le premier développement d'un produit des mises à jour venant après sa mise en production. On parle de phase de maintenance pour les travaux postérieurs au premier développement et, bien souvent, les équipes, les procédures et les outils sont différents.

Avec Scrum, cette distinction n'existe pas : les mises à jour sont produites par les releases successives. Au cours de ces releases, c'est toujours le cadre Scrum et les pratiques agiles qui sont appliqués, le même backlog qui continue de vivre, et de préférence les mêmes équipes qui développent.

Cependant, s'il y a moins de travail à faire, la taille de l'équipe de départ va diminuer. Parfois, on transfère la maintenance vers une équipe spécialisée qui s'occupe de plusieurs projets, voire de plusieurs clients. Un point-clé de cette organisation est la gestion des bugs et demandes d'évolution. Comme ils arrivent par flux, une option est de passer à Kanban.

19.3.2. Gestion des bugs

La notion de bug varie selon les projets et aussi selon les points de vue. Ce que j'ai présenté comme une correction de bug dans le chapitre 6 *Structurer le backlog* ne correspond pas à l'idée que tout le monde s'en fait.

Sur une story en cours

Un défaut trouvé sur une story en cours de développement dans le sprint est une condition d'acceptation en échec : la story n'est pas finie. L'essaim qui s'occupe de la story ajoute simplement la tâche nécessaire pour le corriger. Cela ne va pas dans le backlog de produit et encore moins dans un outil de « *bugtracking* », ce serait de la perte de temps et d'énergie. Ce n'est donc pas considéré comme un bug.

Sur une story considérée comme finie

On peut trouver un bug sur une story développée dans un sprint précédent et qui a été considérée comme finie. À tort, mais c'est la vie. Rentrent aussi dans cette rubrique les

bugs qui portent sur des parties développées avant que le projet mette en place un processus agile.

Un bug enlève de l'utilité à une story, plus ou moins selon sa gravité :

- Un bug critique empêche le fonctionnement d'une ou plusieurs *stories*, dont l'utilité devient nulle.
- Un bug majeur ne permet pas un fonctionnement normal et fait perdre une grande partie de l'utilité à la story.
- Un bug mineur fait perdre un peu de valeur à une story en rendant son utilisation plus difficile.

En principe, si les conditions d'acceptation ont été vérifiées, un bug critique ou majeur ne peut être dû qu'à une régression et nécessite une correction immédiate, sans oublier d'analyser ses causes. Sans passer par le backlog. Le corollaire est que si on découvre qu'il manque une condition d'acceptation à une story, cela est une nouvelle *user story* et non pas un bug. Ne devraient donc être présents dans un backlog que des bugs mineurs, qui ne nécessitent pas d'ajouter une condition d'acceptation.

Les bugs qui sont collectés dans le backlog suivent le workflow de la story : ils sont affinés et ordonnés. C'est au Product Owner de décider si la correction d'un bug (non critique) est plus importante que le développement d'une nouvelle *user story*.

Bug sur du code existant

Si on a adopté une approche de pilotage par les tests, on va d'abord écrire un test unitaire qui va reproduire le bug (test « rouge »), puis on écrit le code juste nécessaire pour faire passer ce test à « vert ». On aura ainsi capitalisé sur le bug et amélioré la qualité globale du logiciel.

19.4. Pratiques de développement sur le terrain

19.4.1. Sortir l'architecte de sa tour d'ivoire

L'architecture évolutive influence la vision du rôle d'architecte. Dans les organisations, on peut identifier deux postures typiques :

- L'architecte qui prend les grandes décisions, mais ne participe pas aux travaux de l'équipe (il reste dans sa tour d'ivoire).
- L'architecte qui montre l'exemple en « mettant les mains dans le cambouis » et en collaborant intensivement avec les autres membres de l'équipe.

Le rôle d'architecte n'existe pas dans une équipe Scrum. La première posture est bannie. La seconde se concrétise dans le rôle d'expert ou, mieux, au sein de l'équipe.

19.4.2. Faire de la conception collective

La réunion de planification de sprint permet d'identifier la nécessité de faire de la conception pour développer une story. Cela fera l'objet d'une conception collective par les

membres de l'essaim qui s'occupe de la story.

Les mêlées quotidiennes permettent d'identifier d'autres besoins de faire de la conception collective.

19.4.3. Essayer les binômes

Il arrive qu'un développeur travaille pendant plusieurs jours sur une tâche, sans réussir à la finir. C'est probablement qu'il est face à un obstacle qu'il n'arrive pas à identifier. Une bonne façon de traiter ce cas est qu'un autre membre de l'équipe se porte volontaire pour une séance de travail en binôme.

19.4.4. Devenir un artisan du logiciel

La meilleure façon pour éviter d'accumuler de la dette technique est de s'améliorer continuellement dans les pratiques de développement.

Pour cela, il faut les considérer comme des pratiques de tout premier plan, ce qui n'est pas toujours le cas de la part des managers ou des Product Owners.

Le mouvement Software Craftsmanship est né pour promouvoir ces idées, qui sont résumées dans le *Manifeste de l'artisanat du logiciel* : <http://manifesto.softwarecraftsmanship.org/#/fr-fr>.

Bien commencer

La question à se poser

Est-ce que tous les membres de l'équipe sont bien formés aux pratiques de développement ?

De mauvais signes

Les développeurs font remonter qu'il y a de la dette technique.

Chacun travaille dans son coin.

Par quoi démarrer

Prendre en main l'environnement de développement, pendant le sprint zéro.

À retenir

L'usage de pratiques d'ingénierie est obligatoire pour une équipe Scrum qui développe un produit logiciel.

Les pratiques de développement venant d'*Extreme Programming* comme l'intégration continue, le développement piloté par les tests et la programmation en binôme s'intègrent bien dans le cadre Scrum.

Avec Scrum, l'équipe fait de l'architecture évolutive et de la conception émergente. L'objectif de ces pratiques est de limiter le nombre de bugs et l'accumulation de dette technique.

Références :

- ☞ Ken Beck, *Test Driven Development by Example*, 2002, <http://agilelib.net/?100157>.
- ☞ Benoît Gantaume, *JUnit, mise en œuvre pour automatiser les tests en Java*, ENI, 2010.

Notes

[1] <http://www.martinfowler.com/articles/continuousIntegration.html>

[2] Le *commit* est le fait, pour un développeur, de mettre le résultat de son travail dans l'espace commun à toute l'équipe. Avec l'avènement de Git, on dit aussi *push*.

[3] Un gizmo ! Voir le billet d'Emmanuel Chenu :
<http://emmanuelchenu.blogspot.fr/2009/04/lintegration-continue-est-un-systeme.html>.

Appliquer Kanban sur Scrum

Scrum est un truc qui paraît tout simple, mais j'en suis quand même à une vingtaine de chapitres pour essayer de l'expliquer. Kanban est un peu pareil, aussi difficile à qualifier. Depuis son lancement il y a quelques années par David Anderson, son positionnement a changé. Au départ, Kanban était considéré comme une méthode agile apportant la notion de « système tiré », issu de Toyota pour le développement de logiciel. Le rapprochement avec Scrum a donné le *ScrumBan* [Ladas, *ScrumBan*] et *Kanban & Scrum, tirer le meilleur des deux* [Kniberg, *Kanban & Scrum*]. En 2009, j'ai participé à la traduction de ce livre et fait des présentations sur ce thème dans les conférences.

Ensuite, Kanban s'est positionné comme une nouvelle méthode pour améliorer les processus de développement. À la fin de ma préface de la deuxième édition du livre de Laurent Morisseau [Morisseau, *Kanban*], je résumais Kanban ainsi :

« pas une méthode agile de plus, mais une nouvelle méthode d'amélioration des processus orientée services, en phase avec les valeurs et principes de l'agilité, qui part de l'existant pour une transition progressive, et qui s'attaque, au-delà des projets, aux organisations, même grandes et même au-delà de l'IT. »

Ma dernière phrase était le reflet de l'orientation actuelle de Kanban, se présentant plus comme une méthode de management voire de gestion du changement, et s'élargissant à tous les domaines, au-delà du développement de logiciel.

Ma volonté en ajoutant ce chapitre n'est pas de proposer Kanban comme alternative à Scrum, mais de montrer comment, dans certaines situations, Kanban peut enrichir Scrum. J'ai déjà commencé dans des chapitres précédents en expliquant par exemple que la représentation du *workflow* de la story avec les bacs et l'essaimage pendant un sprint sont des pratiques issues de Kanban. Dans ce chapitre, je vais pousser un peu plus loin l'idée de Kanban comme méthode d'amélioration appliquée à Scrum.

L'objectif est de répondre à des problèmes courants qui se présentent dans certaines situations avec Scrum.

20.1. Pourquoi Kanban sur Scrum ?

20.1.1. Problèmes rencontrés avec Scrum dans certaines situations

Je m'adresse ici à des équipes déjà expérimentées avec Scrum, qui l'ont mis en place avec succès. En enquêtant sur le terrain, j'ai collecté des problèmes parfois rencontrés. Je les

présente avec une carte des impacts.

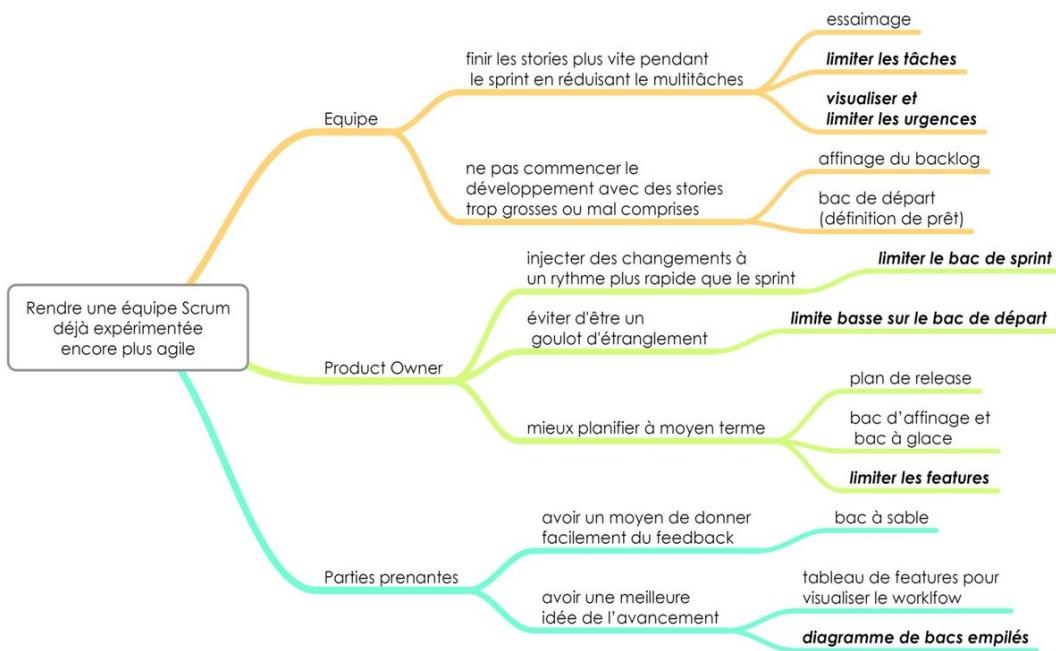


Fig. 20.1 La carte des impacts pour améliorer Scrum

Le but de ces équipes est d'être encore plus agiles. Les impacts liés aux rôles constituent les réponses attendues aux problèmes rencontrés.

Des solutions pour obtenir ces impacts ont déjà été présentées dans les chapitres précédents : essaimage, affinage, etc. Figurent en **gras** (figure 20.1) les autres pratiques Kanban, qui vont être abordées dans ce chapitre.

Les situations principalement concernées sont celles où, parmi les critères de contextualisation présentés dans le chapitre 13 *Contextualiser Scrum*, le taux de changement est élevé et, dans une moindre mesure, le déploiement est fréquent. On constate aussi un impact sur la gouvernance, avec la pratique connue sous l'appellation *Kanban portfolio*.

20.1.2. Brève introduction à Kanban

Kanban est une méthode d'amélioration qui se présente comme du changement progressif et en douceur :

1. Commencer là où on en est.
2. S'engager à changer de manière incrémentale.
3. Respecter initialement le processus actuel, les rôles et responsabilités.

Dans notre approche « Appliquer Kanban sur Scrum », nous partirons donc de Scrum, avec ses sprints, ses événements, ses rôles et son backlog.

Kanban propose six pratiques :

1. Visualiser le workflow.

2. Limiter le travail en cours.
3. Mesurer et gérer le flux de travail.
4. Rendre explicites les règles de gestion du processus.
5. Implémenter des boucles de feedback.
6. S'améliorer de manière collaborative.

L'ordre de présentation correspond à la profondeur d'application de Kanban.

La première est donc la pratique Visualiser, que Scrum promeut également, avec les tableaux.

Nous allons nous concentrer sur la deuxième « Limiter le travail en cours », la pratique essentielle de Kanban.

On peut noter que les trois suivantes sont aussi traitées par Scrum.

20.1.3. Limites Kanban

Dans un « système tiré » à la Kanban, les limites sont utilisées pour rendre le flux plus prévisible. Le but est d'éliminer la surcharge et de créer de la disponibilité.

L'analogie fréquemment utilisée est celle du trafic routier, dans lequel on limite la vitesse pour fluidifier la circulation. C'est plus déroutant dans le monde du travail, où on n'a pas souvent le réflexe de ralentir pour améliorer la performance globale.

La limite Kanban est une pratique consistant à limiter le travail en cours.

Concrètement, la limite est associée à une colonne ou à un bac. Une fois le nombre d'éléments atteint, cela bloque une nouvelle entrée, obligeant à se poser des questions sur quoi faire. Il s'agit ici de la limite haute. Elle sert à éviter les goulets d'étranglement, à avoir trop de stock inutile, à encourager l'entraide.

Il existe une autre limite, moins connue, la limite basse, qui déclenche un réapprovisionnement. Elle apporte une alternative subtile au rythme Scrum.

20.1.4. On garde les sprints

Ce livre portant sur Scrum, ce qui est présenté ici reste du Scrum : on garde le cadre et en particulier les sprints. Éventuellement, une pratique Scrum sera adaptée pour cette application de Kanban, la planification de sprint.

La pratique Kanban introduite dans Scrum est la limitation explicite du travail. Les éléments de travail identifiés dans Scrum sont les tâches, les stories et les features.

20.2. Limiter les tâches

Commençons par appliquer une limite haute sur l'élément le plus fin de Scrum : les tâches.

20.2.1. Limite sur le nombre de tâches en cours

Dans un tableau Scrum pour le sprint, les tâches sont réparties sur trois colonnes : à faire, en cours ou à finir, fini (ajouter, sous prétexte de Kanban, une autre colonne serait une très mauvaise idée).

En observant des tableaux on s'aperçoit que, souvent, il y a beaucoup de tâches dans la colonne du milieu, en cours. Il y en a plus que de personnes dans l'équipe. Parfois un seul développeur en a plus d'une demi-douzaine en cours ! Bien sûr, certaines sont parfois bloquées, en attendant la résolution d'un obstacle. Néanmoins, dans certaines circonstances, il y a manifestement trop de tâches en cours.

Une limite explicite sur les tâches en cours permet de créer des points de discussion et, probablement, d'améliorer les choses. Cependant, les tâches étant associées à des stories, il est préférable d'agir directement au niveau story.

20.2.2. Visualiser et limiter les urgences

Il arrive que Scrum s'avère trop difficile à suivre sur l'aspect « l'équipe n'est pas perturbée pendant le sprint ». Des organisations qui sont passées à Scrum n'y arrivent pas. Il y a toujours des perturbations et la conséquence est que les sprints sont souvent des échecs : l'objectif du sprint, même implicite, n'est pas atteint.

La proposition est d'assouplir la règle de non-perturbation en traitant les urgences comme un flux de changements.

On ajoute une ligne pour les urgences dans le tableau des tâches, en limitant explicitement le nombre de ces tâches urgentes.



Fig. 20.2 Une ligne pour les urgences

Un problème induit par les perturbations est la remise en cause de l'objectif du sprint. Chaque fois qu'une urgence est traitée, il convient de se demander en équipe si cet objectif doit être ajusté. Si l'urgence a pris un temps significatif à l'équipe, c'est tout à fait logique de le reconsidérer. Or, revoir l'objectif du sprint a des implications sur les parties prenantes, ce qui n'est pas anodin.

La limite permet de restreindre le risque d'un ré-ajustement continual. Cependant elle n'est pas évidente à appliquer : la personne à l'origine de l'urgence acceptera difficilement qu'on lui dise « non pas tout de suite ».

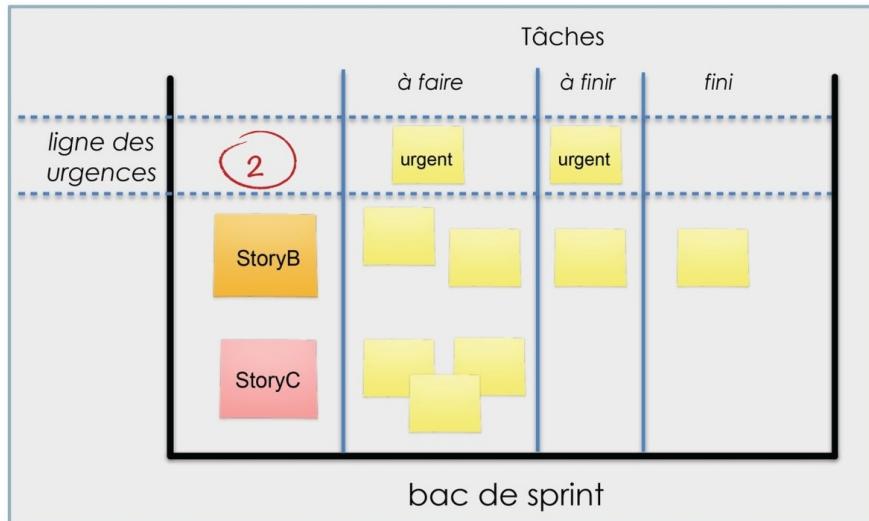


Fig. 20.3 Urgences sur le tableau

Est-ce qu'on limite le nombre d'urgences en tout ou seulement les urgences en cours ? Ou les deux ? L'équipe qui essaie cette technique doit le définir et proposer un chiffre pour démarrer. On rappelle que les limites sont fixées de façon empirique, et ont vocation à baisser. L'équipe doit aussi se mettre d'accord sur la façon de prendre en compte une urgence : immédiatement ou quand une tâche est finie ou quand une story est finie. Dans le but d'éviter les changements de contexte, on préférera ne pas interrompre un développeur dans son travail.

La visualisation des tâches urgentes avec la mise en place d'une limite peut être essayée, et contribue à diminuer les perturbations pendant un sprint. Cependant elle est délicate pour l'équipe, qui risque d'être perturbée gravement. Elle ne devrait être utilisée qu'à titre provisoire.

Rappelons qu'une autre possibilité est de créer une story de réserve, sans contenu, et de l'ajouter dans chaque sprint afin d'y placer les urgences qui surviendront.

20.3. Limiter les stories

20.3.1. Limite dans le sprint

Le lecteur aura préalablement pris connaissance du chapitre 9 *Planifier le sprint*.

Mieux réagir aux changements

Des équipes ont acquis une bonne maîtrise de Scrum, elles respectent le sprint, mais sont parfois frustrées de ne pas pouvoir injecter des changements « urgents » à un rythme plus rapide que le sprint.

Nous avons vu que, lors de la planification du sprint, à la fin, les stories sélectionnées passent « en cours ». Cependant, le nombre de stories sur lesquelles l'équipe travaille est

moindre, en particulier si on pratique l'essaimage.

L'équipe Peetic prend en moyenne 10 stories dans le sprint mais ne travaille que sur 3 à la fois. Au début du sprint, il y en a 7 qui attendent.

On se dit qu'elles pourraient attendre en dehors du sprint, pour en laisser passer d'autres, disons plus urgentes, qui pourraient être découvertes après le début du sprint.

Cela permettrait de mieux réagir aux changements pendant le sprint.

Impact sur la planification de sprint

On conserve toujours Scrum et la notion de sprint. La pratique Kanban consiste à ajouter une limite explicite sur le nombre de stories « en course ». Dans le Scrum normal, la limite est implicite, basée sur la durée du sprint.

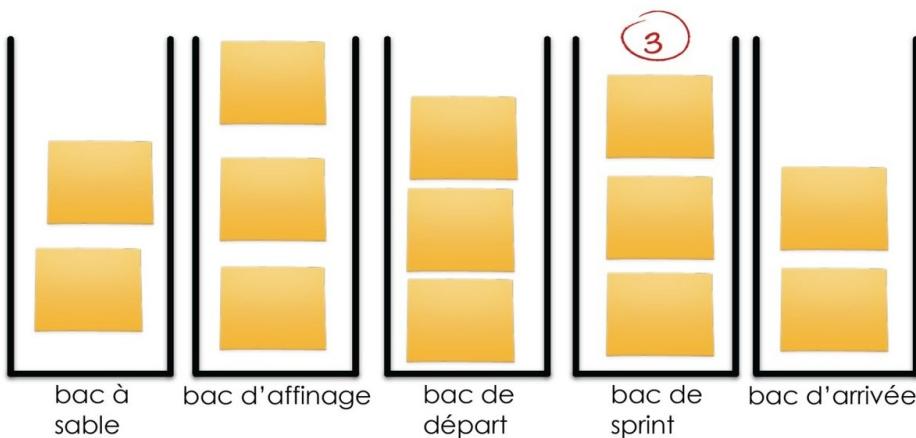


Fig. 20.4 Limite sur les stories du sprint

Cela modifie la façon de mener la planification de sprint : la réunion s'arrête quand la limite est atteinte et la planification se refera pendant le sprint pour réapprovisionner l'équipe.

Cette pratique est tout à fait compatible avec l'essaimage. La limite correspond au nombre d'essaims simultanés que s'autorise l'équipe.

On ne remet pas en question non plus la notion d'objectif du sprint, c'est fondamental pour l'engagement. Cela veut dire que, parmi les stories prêtes qui ne rentrent pas dans le sprint, on pense que les premières seront réalisées.

Pendant le sprint, des séances de planification complémentaires auront lieu pour ajouter des stories. Elles se tiendront après une mêlée qui aura mis en évidence le besoin de réapprovisionner.

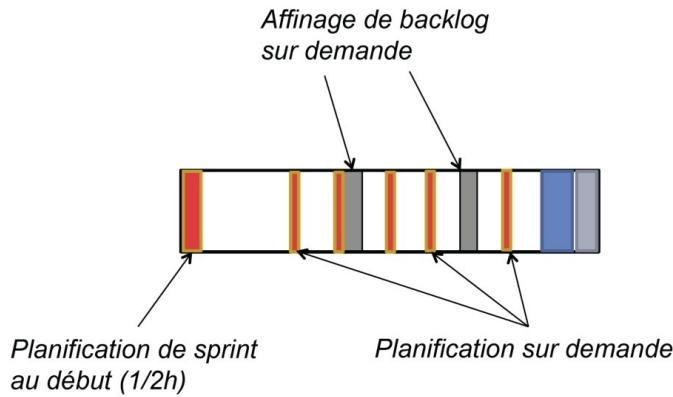


Fig. 20.5 Les événements du sprint revisités

Bénéfices

Le bénéfice principal est une meilleure adaptation au changement.

La limite sur les stories en cours renforce l'agilité.

Les indicateurs de sprint comme le burndown chart deviennent inutiles.

Il est plus facile de construire le tableau de sprint physique. Il y a besoin de moins de place pour les stories en cours et leurs tâches, et leur nombre est stable de sprint en sprint (on ne change pas la limite si souvent).

L'équipe Peetic a une limite de 3, elle peut ne laisser que 3 places pour les stories du sprint, dont une réservée à une « story technique ».

Risques

La planification échelonnée rend plus difficile de définir l'objectif du sprint et de s'y tenir.



Fig. 20.6 Développeur emporté par le flot

L'ajout de stories pendant le sprint est risqué si la définition de prêt n'est pas bien établie ou si elle est mal vérifiée.

20.3.2. Limite dans le backlog

Pour aller plus loin, on peut étendre le principe en restreignant la taille des bacs. Dans les chapitres sur le backlog, j'ai donné des indications sur la taille de chaque bac.

La pratique des bacs peut déjà être considérée comme une visualisation du workflow, la première pratique de Kanban. Aller plus loin en limitant les bacs permettrait de réduire encore plus le stock.

Limites sur le bac de départ

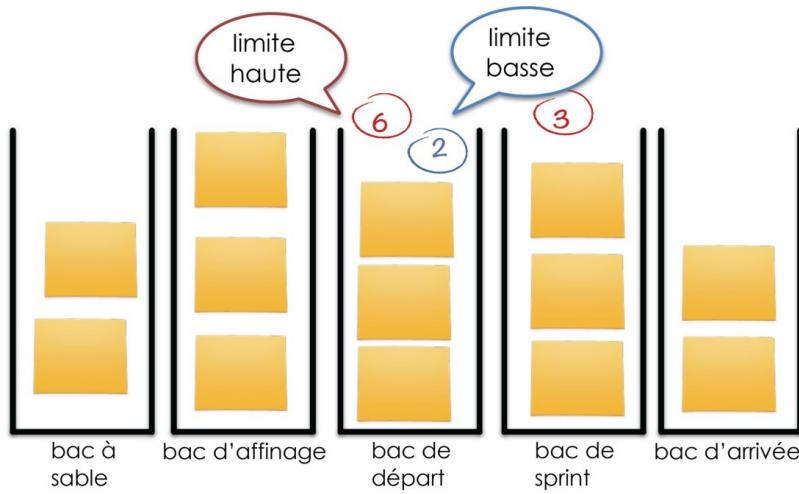


Fig. 20.7 Limites sur bac de départ

Une limite haute permet de limiter le nombre de stories prêtes. Nous avons vu qu'il était bien d'avoir un peu de marge en stories prêtes, par exemple 20 à 50 % de plus que ce qui est fini habituellement en un sprint.

Dans le cas de limite sur le sprint, ce pourcentage s'applique sur ce nombre.

Une limite basse sur les stories prêtes permet de déclencher une séance d'affinage pour réapprovisionnement.

Si la limite haute provoque une discussion quand elle est atteinte, l'arrivée d'une colonne à sa limite basse est le signe d'un besoin d'ajouter des éléments, pour éviter une disette dans les activités suivantes.

Limite sur le bac d'affinage

La pratique de la planification de release et l'usage du bac à glace sont déjà un moyen de limiter la taille du bac d'affinage.

Dans ce cas, il ne sert pas à grand-chose de limiter la taille du bac d'affinage, si ce n'est que cela aide à ne pas décomposer trop tôt des stories épiques.

Une limite se justifie plus dans le cas où le bac d'affinage n'est pas déjà contraint par la release.

J'ai rencontré des situations où on stockait des centaines d'éléments, parfois pendant très longtemps.

Introduire une limite permet de commencer à réduire la taille de ces backlogs. En régime normal, une limite de deux fois celle sur les stories prêtes est raisonnable.

Le bac à sable étant destiné à recueillir les demandes de tout le monde, il n'est pas recommandé de le limiter, ce serait une entrave à l'expression.

20.4. Limiter les features

Limiter les stories c'est bien, limiter les features c'est mieux.

20.4.1. Limite sur les colonnes

Les features sont placées dans un tableau dont les colonnes représentent le *workflow*.

Ajouter une limite sur une colonne pousse à diminuer le stock, c'est-à-dire des features commencées et pas encore finies. La friction apportée par la limite entraîne une discussion sur ce qui est prioritaire : commencer ou finir ?

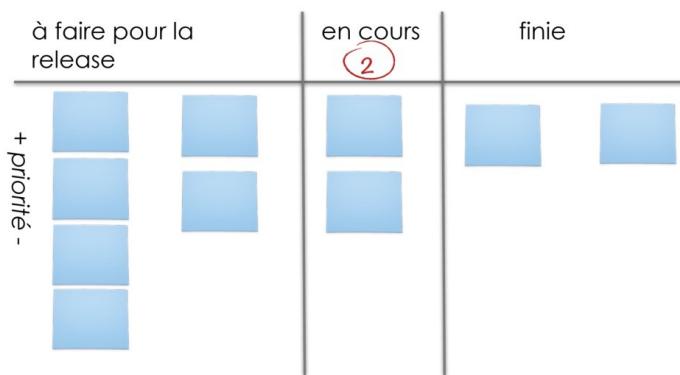


Fig. 20.8 Limite sur les features

Dans le cas où la feature est déployée auprès des utilisateurs finaux dès qu'elle est finie, les limites vont permettre de fluidifier le processus, diminuant ainsi ce qu'on appelle le temps de cycle.

Le tableau de features, notamment quand il sert à plusieurs équipes, est l'équivalent de ce qu'on appelle le *Kanban portfolio*, et facilite la planification stratégique.

20.4.2. Et sur les impacts ?

C'est bien de fournir de la valeur avec des features, c'est encore mieux de s'assurer que cela a un impact réel sur les acteurs.

Les organisations qui utilisent l'*impact mapping* pour la planification stratégique appliquent implicitement du Kanban en ne traitant qu'un impact à la fois (plus le feedback qui remonte des précédents).

Il est souhaitable de mener les réflexions sur les limites selon cette hiérarchie : impact, feature, story puis tâche.

20.5. Mesures et indicateurs

20.5.1. Temps de traversée et temps de cycle

Kanban vient avec d'autres types de mesures.

Le **temps de traversée** est la durée que met un élément à parcourir la chaîne complète, à partir de la formulation de la demande jusqu'à son utilisation en production. Dans notre contexte, cela n'aurait un sens que pour des features, à condition qu'on les déploie une fois finies.

C'est facile à mesurer, si on a ajouté la date au bon moment sur la feature.

En revanche, produire un indicateur utilisant le temps de traversée, comme une carte de contrôle, ne me paraît pas pertinent, compte tenu du faible nombre de features.

Le **temps de cycle** est la durée que met un élément à parcourir le workflow ou une partie, qui sera celle sur laquelle une équipe pourra s'engager une fois que les mesures analysées montreront une stabilité.

L'équipe s'engageant déjà sur le sprint, mesurer le temps de cycle sur des stories n'est pas utile.

20.5.2. Diagramme de bac empilé

Un indicateur spécifique de Kanban est le diagramme de flux empilé. Il montre le nombre d'éléments par colonne. Pour les stories, ce serait par bac, et cela donne le diagramme de bac cumulé.

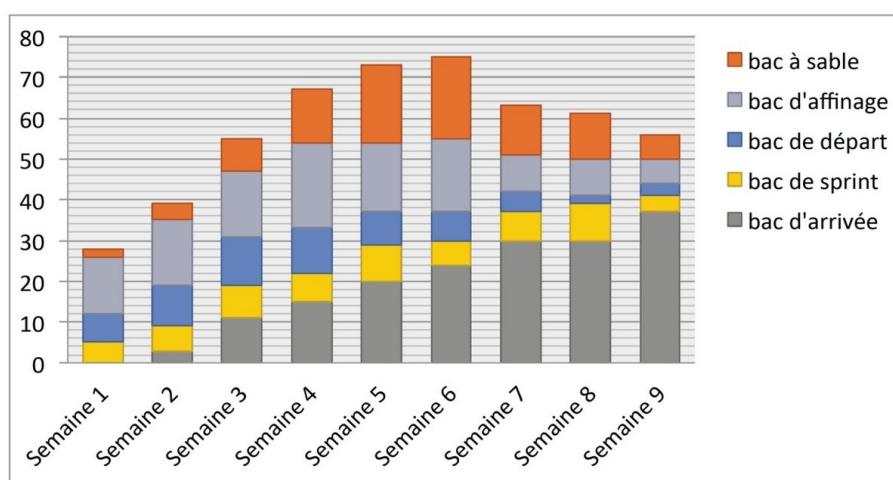


Fig. 20.9 Un diagramme de bacs empilés

Usage	Ce diagramme permet au Product Owner d'ajuster le contenu d'une release en voyant l'état des stocks dans les bacs. Il permet surtout une analyse du pourquoi a posteriori.
Quand l'utiliser	Dès qu'on met en place le système des bacs.

Collecte	Le nombre de stories dans chaque bac en commençant par la droite (bac d'arrivée) et en cumulant.
Unité	En nombre de stories.
Tendance souhaitée	Pas d'à-coups dans le flux (et avoir vidé le bac d'affinage à la fin de la release).

Je conseille de le mettre à jour non pas à chaque sprint, mais sur un rythme hebdomadaire.

On me dit souvent qu'il est difficile à mettre en place et je concède bien volontiers qu'il demande quelques efforts, mais pas si grands une fois qu'on a mis en place les bacs. Parfois, on me dit aussi qu'il est difficilement interprétable. En fait, c'est un outil d'analyse globale et il faut avoir suffisamment de données. À ce moment-là, l'effort fait pour l'étudier est récompensé par la richesse des informations qu'on en tire.

20.6. Arrêter Scrum pour Kanban ?

Ce n'est pas l'objet premier de ce chapitre, qui est d'appliquer Kanban en gardant Scrum. Mais il y a des risques de mal l'appliquer. Il est aussi légitime de se poser la question d'arrêter Scrum et de ne garder que Kanban. Ce n'est pas un problème en soi, il peut y avoir de bonnes raisons à cela. Mais il peut y en avoir aussi de mauvaises.

20.6.1. Risques dans l'application de Kanban sur Scrum

Un risque, déjà évoqué, serait de se contenter d'appliquer les limites sur des tâches. Nous avons aussi parlé du risque principal en préambule : essayer cette application de Kanban alors que l'équipe ne pratique pas déjà Scrum avec succès. Il y a de fortes chances qu'elle ne fasse alors ni du Scrum ni du Kanban.

Kanban, ce n'est pas juste des colonnes avec des Post-it® dedans.

20.6.2. Mauvaises raisons d'arrêter Scrum pour Kanban

Parmi les raisons évoquées par ceux qui disent « on arrête Scrum pour passer à Kanban », certaines m'apparaissent très discutables :

- les réunions prennent trop de temps,
- le sprint est un carcan qui met la pression sur les équipes,
- on déploie à un rythme différent du sprint, alors pourquoi le garder ?

À cela, on peut répondre que ce ne sont pas des « réunions », et que si on y regarde de plus près, ce sont souvent les estimations qui prennent du temps et qu'on peut tout à fait les supprimer en continuant Scrum.

La pression sur les équipes vient bien souvent de l'engagement de sprint mal compris, et cela peut s'améliorer.

Déployer très souvent c'est bien, mais cela ne remet pas en cause le principe de boîte de temps du sprint ; le rythme régulier garde des vertus sans le déploiement de fin de sprint, par exemple pour fixer des points de rendez-vous avec les parties prenantes.

En outre, Kanban est, de mon point de vue, plus difficile à bien appliquer que Scrum. Il demande de l'expertise en statistiques que bien peu de développeurs possèdent.

20.6.3. Bonnes raisons d'arrêter Scrum

Trop d'urgences

Comme nous l'avons dit dans le premier chapitre, Scrum n'est pas adapté à toutes les situations, en particulier aux changements continuels. Comment savoir si le contexte d'une équipe convient ou pas ?

Essayez Scrum sur plusieurs sprints.

Calculez votre degré d'urgence : sur ce qui est fini dans un sprint, faites la part de ce qui était inconnu au début du sprint. Si, sur plusieurs sprints, ce pourcentage ne diminue pas et reste supérieur à 40 % (c'est empirique), il vaut mieux arrêter les sprints et donc Scrum.

Être plus agile

Une équipe Scrum qui a mis en place avec succès l'application de Kanban telle que nous venons de la présenter peut s'interroger sur l'arrêt des sprints.

Elle maîtrise les limites et gère bien le flux de stories.

Elle se sent suffisamment mûre pour désynchroniser la planification, la revue et la rétrospective. Les faire sur demande, en cas de besoin, plutôt que sur le rythme du sprint.

Quand on arrête les sprints, on peut dire que ce n'est plus du Scrum. Ce n'est pas grave, l'essentiel est d'être agile.

Bien commencer

La question à se poser

Est-ce que nous nous maîtrisons déjà bien Scrum avant de vouloir appliquer Kanban ?

De mauvais signes

Une colonne en plus dans le tableau de sprint.

L'équipe dit faire du Kanban mais il n'y a pas de limites.

Par quoi démarrer

Commencer par l'essaimage.

Kanban est considéré dans ce chapitre comme une méthode d'amélioration de processus. C'est comme cela

Une lecture pour tous

qu'il est présenté dans le livre de Laurent Morisseau [Morisseau, *Kanban*].

À retenir

Appliquer Kanban sur Scrum permet aux équipes ayant déjà réussi avec Scrum d'être encore plus réactives aux changements.

La pratique Kanban appliquée sur Scrum est la limite du travail en cours. On peut limiter le nombre de tâches, mais il est préférable de s'attaquer aux stories ou aux features.

Cela reste du Scrum et en renforce l'agilité en s'alignant avec ce slogan :

« *Arrêter de commencer, commencer à finir.* »

Références :

☞ Henrik Kniberg et Mattias Skärin, *Kanban et Scrum tirer le meilleur des deux*, VF, 2010.

<http://www.aubryconseil.com/blog/public/prez/old/KanbanAndScrum-FRv2.pdf>

☞ Henrik Kniberg, *Lean depuis les tranchées*, VF draft 0.9, 2011.

http://www.aubryconseil.com/blog/public/prez/old/Lean_depuis_les_tranchees.v1.pdf

☞ Corey Ladas, *ScrumBan*, 2009.

<http://agilelib.net/?100009>

☞ Laurent Morisseau, *Kanban pour l'IT*, Dunod, 2014.

Développer un produit avec plusieurs équipes

Le chapitre « Scrum à grande échelle » avait été ajouté lors de la deuxième édition de ce livre. Depuis, le sujet a pris de l'ampleur, des *frameworks* ont été proposés, « commercialisés », c'est-à-dire accompagnés de leurs inévitables certifications, et comparés dans les conférences.

De mon côté, j'ai expérimenté ce passage à l'échelle dans plusieurs situations. Et ma position a changé : je suis revenu à plus de simplicité, plus de Scrum, dirais-je. À mon sens, le Scrum à grande échelle doit rester dans l'esprit, par exemple en évitant de créer de nouveaux rôles.

Pour être plus clair sur l'objectif de ce chapitre, je l'ai renommé. En effet, le *scaling Scrum* est devenu un sujet fourre-tout, qui regroupe des préoccupations bien différentes. Dans mon livre, j'ai choisi de différencier le Scrum pour développer à plusieurs équipes, qui fait l'objet de ce chapitre, de l'agilité pour transformer les organisations, qui sera abordée dans le chapitre suivant.

21.1. Un projet Scrum ?

Le sujet concerne ce qu'on appelle volontiers les « grands projets », souvent avec l'idée sous-jacente que l'agilité n'est pas faite pour ça.

Avant de l'aborder, il nous faut revenir sur la notion de « projet ». La définition classique^[1] est la suivante :

« *Un projet est un effort temporaire dans le but de créer un produit, un service ou un résultat unique.* »

Avec Scrum, l'effort du sprint permet d'obtenir un incrément de produit et, après les efforts de plusieurs sprints, le résultat correspond au produit. L'équivalent le plus proche de la notion de projet serait, avec Scrum, la release, vue comme une suite de sprints.

On pourrait donc assimiler, en Scrum, un projet à l'effort d'une équipe qui développe une version d'un produit, pendant la durée d'une release.

Mais, après la release, il y en a d'autres, et cela continue pendant toute la vie du produit.

Scrum ne correspond pas à la définition du projet : ce n'est pas un effort temporaire, ce n'est pas pour produire un résultat unique.

La notion de projet ne convient pas pour Scrum.

Scrum est fait pour développer des produits : une longue durée est déjà couverte avec les releases successives. Mais comment Scrum peut-il s'appliquer quand les autres dimensions d'un projet classique (périmètre, taille) grandissent, par exemple dans les situations suivantes :

- Quand le nombre de personnes nécessaires va au-delà de la limite Scrum pour une équipe, et qu'il faut donc plusieurs équipes.
- Quand le périmètre concerné est bien supérieur à quelques *stories* et qu'il porte sur plusieurs centaines.

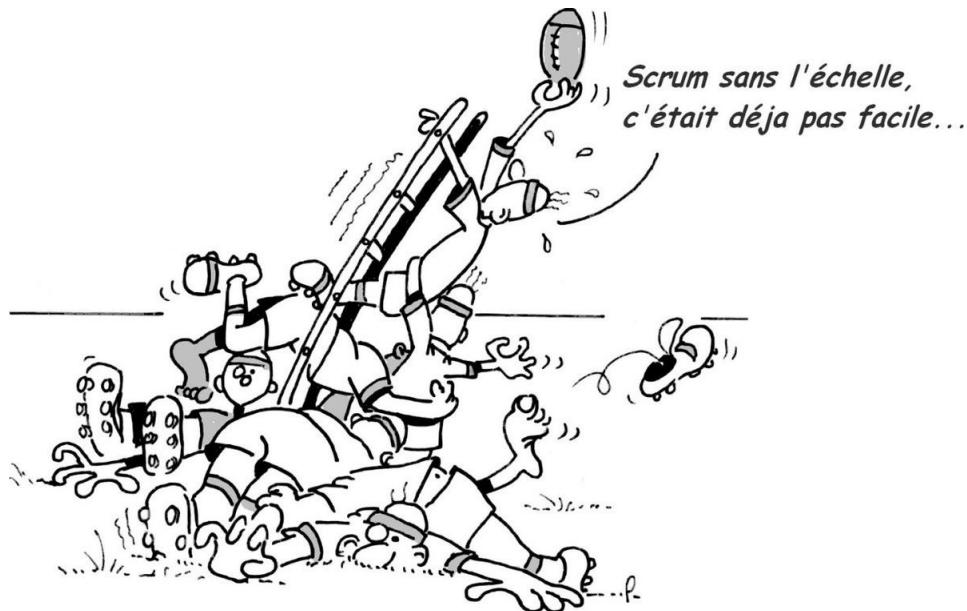


Fig. 21.1 Scrum à l'échelle.

Nous allons examiner les impacts de ce changement d'échelle sur les différents éléments du cadre Scrum.

Attention, avant d'espérer tirer des bénéfices du Scrum à plusieurs équipes, il faut déjà bien maîtriser le Scrum à une équipe.

21.2. Cycle de vie produit

Le lecteur aura préalablement pris connaissance du chapitre 2 *Le cycle des sprints*.

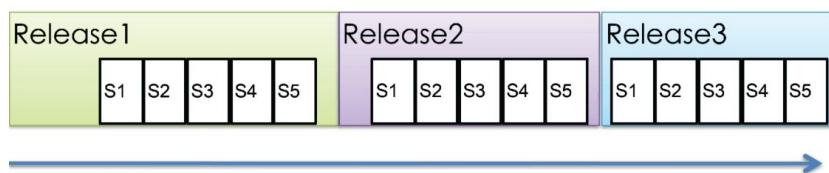


Fig. 21.2 Releases successives dans la vie d'un produit.

Avec Scrum, pas de phase de premier développement suivie d'une phase de maintenance : la vie du produit est constituée de releases successives.

21.2.1. Synchronisation des équipes

Quand plusieurs équipes développent le même produit, chacune déroule son sprint à un moment donné. Ce qui change, c'est qu'une équipe ne définit pas son propre rythme, avec une durée de sprint qui lui serait spécifique.

Toutes les équipes travaillent au même rythme : toutes démarrent et finissent les sprints en même temps.

Exemple : des releases tous les trois mois, des sprints de deux semaines, cinq sprints dans une release.

L'intérêt fondamental de cette pratique est de renforcer la fréquence de l'intégration pour toutes les équipes. On n'attend pas la livraison de l'équipe la plus lente pour mettre tous les morceaux ensemble. L'intégration fréquente améliore la qualité et réduit les risques. Le feedback peut être remonté plus rapidement, ce qui limite le coût d'un changement.

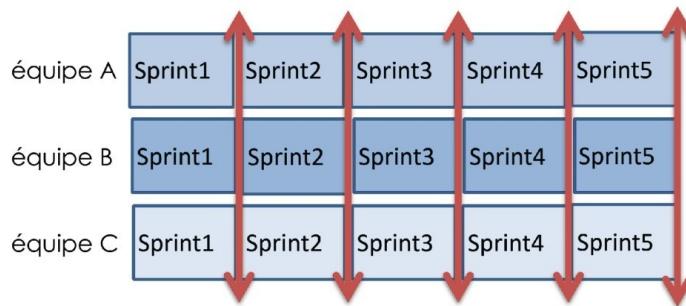


Fig. 21.3 Sprints synchronisés.

21.2.2. Releases à date fixée

De la même façon que les sprints sont synchronisés, les releases des équipes le sont aussi : même date de début, même date de fin, tout le monde suit le même rythme.

L'objectif de chaque release, commun à l'ensemble des équipes, est alors d'apporter le plus de valeur possible dans le temps imparti et connu à l'avance. C'est l'ajustement du périmètre fonctionnel qui permet de maximiser la valeur, avec la date et le coût fixés à l'avance.

On parle de « train de release » que prennent toutes les équipes.

Deux points sont à noter pour la fin de release, à plusieurs équipes :

- Elle fait l'objet d'une synchronisation poussée entre les équipes, qui passent du temps ensemble, pour la rétrospective et la planification.
- C'est à ce moment que la constitution des équipes peut évoluer.

21.3. Les gens avec plusieurs équipes

Le lecteur aura préalablement pris connaissance des chapitres : 3 *Les gens de Scrum*, 4 *Le rôle du Product Owner*, 5 *Le rôle du ScrumMaster*.

21.3.1. Équipes « features »

Quand plusieurs équipes participent à la réalisation du même produit, comment les organiser ? La pratique de partage du travail recommandée est connue sous l'appellation « équipe feature »[Larman, *Feature team*].

Nous avons longuement présenté la notion de feature, ce qui facilite la compréhension du concept :

Les équipes sont organisées de telle sorte qu'elles soient capables de développer une feature de bout en bout.

Attention, cette façon d'organiser les équipes est un changement, voire un choc organisationnel, dans la mesure où la pratique habituelle du partage du travail, hors agile, est plutôt basée sur l'architecture du système. L'enjeu est alors de passer d'équipes « composant » à des équipes « feature ».

Une « équipe feature » est une équipe Scrum, avec ses caractéristiques habituelles : autogérée, pluridisciplinaire et stable. La pluridisciplinarité d'une équipe revêt une importance cruciale, pour fournir de la valeur en limitant les dépendances avec d'autres équipes.

Chaque « équipe feature » possède son propre backlog (de stories). Elle a un ScrumMaster qui lui est dédié. Pour le rôle de PO, c'est différent.

21.3.2. Le rôle de Product Owner

Le rôle de Product Owner a été défini comme associé à un produit et à une équipe. Avec la mise à l'échelle, il y a plusieurs équipes et toujours un seul produit, cela impacte donc le rôle.

Le plus simple serait de conserver l'association entre le rôle et le produit (d'où vient son nom) : un seul PO pour le produit, qui intervient pour plusieurs équipes. Cela fonctionne si le nombre d'équipes reste limité, jusqu'à trois, voire quatre si ce sont de petites équipes.

Au-delà, il devient nécessaire d'avoir d'autres PO, par exemple un pour chaque grand domaine fonctionnel du produit. Il est aussi possible de conserver un PO par équipe, même à deux ou trois équipes. Nous arrivons donc très vite à avoir un groupe de PO pour un produit.

Ce groupe de PO fonctionne en équipe Scrum dans le sens où il aura à donner la vision et à prioriser les features. Il lui faut un PO.

Le groupe pourra désigner un d'entre eux comme ce « PO de PO ». Beaucoup de décisions se prennent de façon collégiale, mais c'est lui qui arbitrera en cas de besoin. PO de ce groupe, ce n'est pas un rôle nouveau. Il peut, éventuellement, continuer à être le PO d'une équipe.

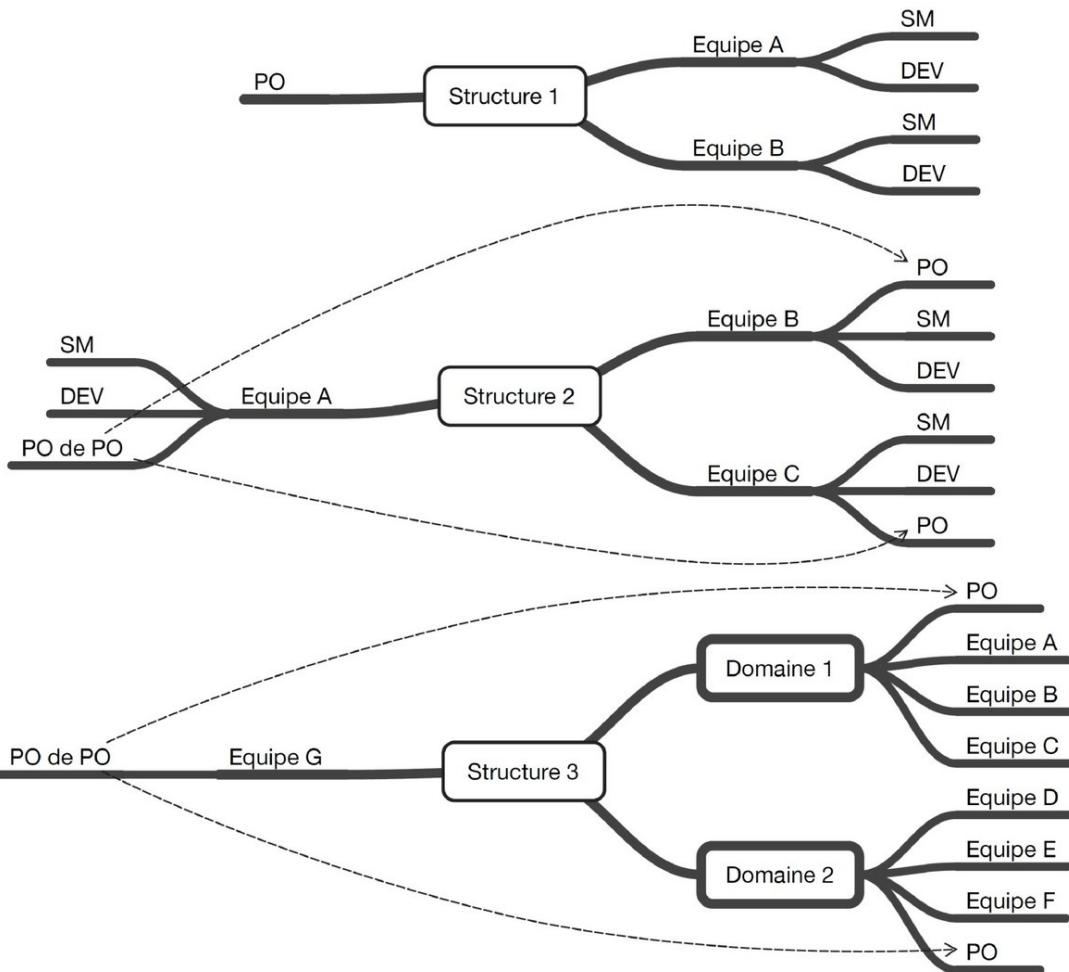


Fig. 21.4 La position du PO dans différentes situations

Exemples d'affectations possibles pour le rôle de PO :

- Structure 1 : 2 équipes et un seul PO.
- Structure 2 : 3 équipes avec chacune son PO, un étant PO de PO.
- Structure 3 : 7 équipes avec 3 PO, un d'entre eux, celui qui s'occupe d'une seule équipe, jouant le rôle de PO de PO.

21.3.3. Passer aux équipes « feature »

La transition d'équipes « composant » à des « équipes feature » n'est pas une mince affaire. Une solution douce est de commencer par créer une seule équipe Scrum, puis d'élargir progressivement à d'autres équipes.

Les premières équipes auront besoin de disposer d'experts en architecture du logiciel ou du système. Ceux-ci seront alors détachés de leur structure initiale pour aider l'équipe, en tant qu'experts, sur des études sur des *sujets techniques*. À plus long terme, ces architectes intégreront des « équipes features », ou auront transféré leur compétence à l'équipe.

L'allocation d'une feature à une des équipes, pour sa réalisation, constitue une activité nouvelle qui sera ajoutée à l'affinage.

Le meilleur moment pour faire évoluer la constitution des équipes se situe en fin de release. Il ne s'agit pas de changer toutes les équipes, car comme nous l'avons vu, la stabilité est un facteur important de réussite. Cependant, il peut y avoir quelques évolutions au sein des équipes. Il peut arriver aussi qu'une nouvelle équipe soit constituée si l'alimentation du backlog le nécessite. Quand le flux des évolutions ralentit, on peut aussi envisager, dans l'autre sens, la fusion d'équipes.

21.4. Backlog et affinage à plusieurs équipes

Le lecteur aura préalablement pris connaissance des chapitres 6 *Structurer le backlog*, 7 *Affiner le backlog* et 8 *La définition de fini*.

21.4.1. Backlog d'équipe avec des stories

La notion clé dans le travail d'une équipe est la story, qui est tirée du backlog pour être réalisée dans un sprint.

S'il y a plusieurs équipes, quelle que soit leur taille, sur quel backlog travaillent-elles ? Des équipes travaillant sur le même produit se partagent-elles un seul backlog ?

Il est important qu'une équipe sache ce qu'elle va faire pendant un sprint. Il est aussi essentiel qu'elle ait une connaissance des stories sur lesquelles elle va travailler dans les sprints suivants. Cela ne serait pas facile avec un backlog partagé entre de nombreuses équipes. C'est pourquoi chaque équipe a son backlog (cependant, avec deux ou trois équipes, il est encore possible d'afficher en une seule vue le bac d'affinage complet).

Il est finalement plus approprié de parler de backlog d'équipe que de backlog de produit.

21.4.2. Tableau de features partagé

Pour un gros produit, on pourrait avoir un backlog qui serait l'agrégat de tous les backlogs d'équipe. Ce n'est pas une très bonne idée : avec toutes les stories de toutes les équipes, cela ferait bien trop d'éléments pour espérer le gérer aisément.

En considérant qu'en moyenne un backlog – d'équipe donc – contient une cinquantaine d'éléments, si un produit est développé avec, par exemple, cinq équipes, cela ferait plus de 200 stories.

Même en utilisant le système de bacs, cela en ferait beaucoup dans le bac d'affinage et on risquerait alors d'en oublier fortuitement, ce qui, pour certains, serait bien embêtant.

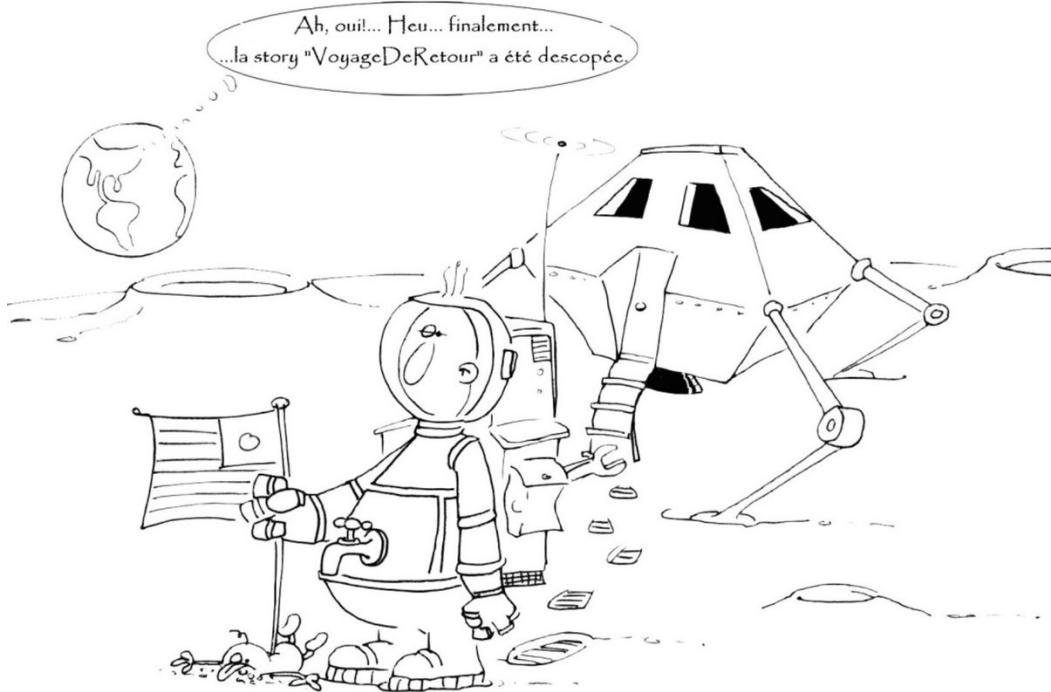


Fig. 21.5 Une story oubliée dans un programme spatial.

Il est donc nécessaire de traiter les problèmes d'échelle à un plus haut niveau que les stories.

Le tableau de features est déjà intéressant pour une seule équipe. Avec plusieurs, il est indispensable. Comme dans le backlog de stories, la notion d'ordre y est fondamentale pour avoir une vue des priorités.

Il permet aussi de séparer le travail entre chaque équipe : une feature est généralement associée à une seule équipe. Un attribut (une couleur par exemple) ajouté à chaque feature indique l'équipe qui en a la charge (figure 21.6). L'équipe qui a la responsabilité d'une feature la décompose en stories qu'elle place dans son backlog d'équipe.

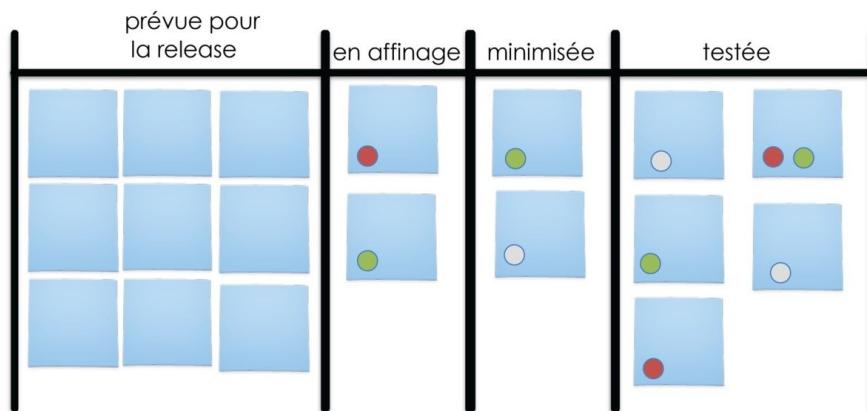


Fig. 21.6 Tableau de features pour trois équipes

Feature épique

Dans les systèmes, qu'ils soient d'information (SI) ou plus techniques, on trouve souvent des projets particuliers qui portent sur plusieurs produits et possèdent souvent une forte composante technique. On les appelle parfois projets transverses, ou chantiers.

Ces travaux seront considérés comme des grosses features, dont le développement peut aller au-delà de la durée d'une release, voire durer plus d'un an. Elles seront donc épiques, dans le sens où il faudra les décomposer avant de les affecter à une équipe « feature ».

Exemples : système de licence, système de gestion de l'identité, changement de système de push, accès web pour des applications, etc.

Après leur étude, quand la décision est prise de les développer, elles sont décomposées en features qui se retrouvent dans le tableau pour être affectées à une équipe, non pas en fonction du domaine cette fois, mais de façon opportuniste.

Nous avons donc un tableau de features unique pour le produit et un backlog de stories par équipe.

21.4.3. Liste des impacts

Le lecteur aura préalablement pris connaissance du chapitre 14 *Découvrir le produit*.

À très grande échelle, le nombre de features peut être important : une équipe en réalisant en moyenne une dizaine par release, le tableau de features peut être chargé au niveau produit à partir d'une demi-douzaine d'équipes.

La planification stratégique du produit peut alors se faire au niveau des impacts. Une feature apporte suffisamment de valeur pour être livrée, alors qu'un *impact* vise à apporter un avantage vraiment compétitif.

Mettre en avant les impacts facilite la planification stratégique et garantit l'alignement. En général, un impact correspond à un domaine fonctionnel et sera donc naturellement associé à un PO.

C'est le groupe de PO qui priorise les impacts.

21.4.4. Structure des éléments de travail

Notre modèle comporte trois niveaux : story, feature, impact.

La vue produit est présentée sur trois niveaux d'éléments, plus les tâches des sprints. Voici un résumé des utilisateurs principaux pour chacun, ainsi que des façons de les estimer.

Tab. 21.1

	Cible	Attribut de valeur
Impact	Management, groupe de PO	Hypothèse vérifiable de valeur financière ou d'utilité
Feature	Groupe de PO, équipe	Valeur relative non vérifiable

Story	Équipe	Très difficile à estimer
Tâche	Essaim d'une équipe	Aucune valeur

Illustration avec les trois niveaux :

- **story** : « en tant que propriétaire d'un basset hound, je peux poser une question canine à un expert ».
- **feature** : coaching de chiens de race en ligne, estimée à une valeur de 17.
- **impact** : 1 000 personnes se sont abonnées au service de coaching.

21.4.5. Affinage à grande échelle

L'affinage traditionnel, fait par chaque équipe, est complété par un affinage de plus haut niveau, sur les features. Il sert à décider quelle nouvelle feature sera réalisée par quelle équipe, à partir du tableau. Les features sont donc associées à une équipe.

Eh bêêêê ! Nous, on sait faire !

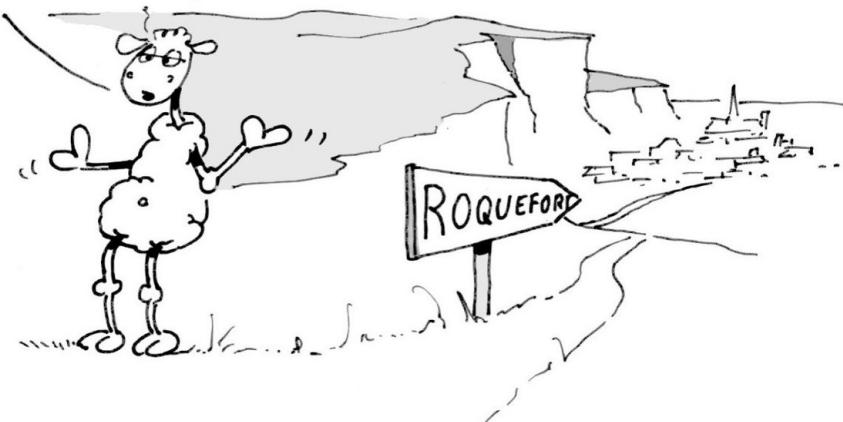


Fig. 21.7 L'affinage à grande échelle est bien connu dans l'Aveyron.

Cette session d'affinage à grande échelle précède la réunion d'affinage de chaque équipe, qui a lieu immédiatement après.

À cette réunion participent le groupe de PO, des experts et des représentants d'équipes (dans le cas où une équipe n'a pas de PO dédié).

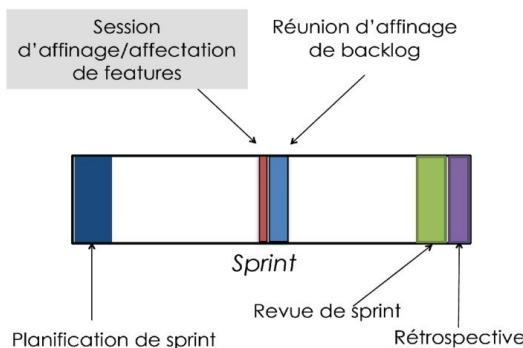


Fig. 21.8 La séance d'affinage global

Les activités suivantes y sont menées :

- décomposer les grosses features (les features épiques),
- discuter des prochaines à faire, identifier les dépendances et les risques pour définir si elles sont prêtes,
- ordonner les features dans les colonnes selon les critères valeur, taille et dépendances,
- identifier l'équipe qui va se charger de la réalisation de la feature.

21.4.6. Définitions de prêt et de fini à l'échelle

Les définitions de prêt et de fini sont uniques pour toutes les équipes, afin d'assurer un niveau de qualité homogène.

Elles fonctionnent de façon identique à ce que nous avons vu pour une seule équipe et sont élaborées collectivement pendant le sprint zéro. Un effort particulier sera mis sur la définition de fini pour une feature.

21.5. Les événements du sprint à l'échelle

Le lecteur aura préalablement pris connaissance des chapitres 9 *Planifier le sprint*, 10 *La mêlée quotidienne*, 11 *La revue de sprint* et 12 *La rétrospective*.

21.5.1. Planification de sprint

Chaque équipe tient sa planification de sprint comme d'habitude, toutes ont lieu en même temps. La seule différence est qu'un PO n'y participe pas toujours. En effet, dans le cas où un PO s'occupe de plusieurs équipes, il ne peut pas se multiplier.

En revanche, une courte réunion globale a lieu juste avant, pour rappeler et éventuellement ajuster la répartition des features, rappeler l'objectif de la release et surtout identifier les besoins de synchronisation entre les équipes. Bien qu'on soit organisé en équipes « feature », il arrive qu'une feature demande la collaboration de plusieurs équipes. Ce point est identifié pendant l'affinage et confirmé pendant cette réunion.

Cette réunion regroupe tout le monde à deux ou trois équipes et, à plus, seulement les personnes qui ont participé à l'affinage global.

21.5.2. Mêlée

À la question comment Scrum gère cette dimension « grands projets », la réponse est souvent :

« *Ah, pas de problème, on fait du scrum de scrums* ».

Le fameux, le mythique *scrum de scrums*. Cela me rappelle l'énorme mêlée des origines du rugby, évoquée par Daniel Herrero [Herrero, *Rugby*], mais là ce n'est pas tout le monde ensemble.

Le principe est le suivant : chaque équipe déroule sa mêlée de façon habituelle ; puis juste après, la mêlée des mêlées a lieu, avec un membre de chaque équipe.

C'est une technique qui facilite la communication entre plusieurs équipes.

Les mêlées des équipes peuvent être légèrement décalées pour qu'une personne puisse éventuellement participer à plusieurs. Elles sont suivies de la « mêlée des mêlées » dont la fréquence peut être relâchée, par exemple tous les deux jours. À cette réunion participent des représentants de chaque équipe, ScrumMasters ou autres, les mieux placés pour discuter des problèmes de communication inter-équipes.

De la même façon, une réunion se tient entre les PO s'il y en a plusieurs.

Au cours de cette mêlée des mêlées peut être relevé un obstacle concernant plusieurs équipes. Une réunion regroupant des représentants de ces équipes se tiendra alors dans la journée, pour l'éliminer.

21.5.3. Revue

La revue est faite en commun avec toutes les équipes qui participent au même produit. Les démonstrations sont faites à partir de l'incrément de produit élaboré conjointement.

Le groupe de PO anime cette réunion qui, en comparaison à la revue d'une seule équipe, prendra évidemment un peu plus de temps car il y a plus de choses à montrer. Les équipes et toutes les parties prenantes sont invitées.

21.5.4. Rétrospective

Chaque équipe déroule sa rétrospective de sprint comme d'habitude. À la demande d'une équipe, on peut organiser une rétrospective globale. Elle peut avoir lieu alors que le sprint suivant est commencé. Une équipe fait cette demande lorsqu'elle fait face à un obstacle d'organisation ou pour faire évoluer la définition de fini commune.

En fin de release, une rétrospective est organisée avec toutes les équipes. Elle revient sur un passé plus lointain et dure plus longtemps. On peut y consacrer une journée et en profiter pour la mener de façon différente. Par exemple, cela peut être l'occasion d'un « open space » ou forum ouvert, auquel sont invités tous les membres d'équipe et les parties prenantes.

21.5.5. Résumé

	Affinage	Planification de sprint	Mêlée	Revue	Rétrospective
Conservé	Une séance par équipe	Planif. de sprint par équipe	Une mêlée par équipe		Une rétro par équipe
En plus	Affinage global avec	Une présentation globale des	Une mêlée pour la synchro.	Une revue globale	Une rétro globale sur

le groupe de PO	affectations de feature	Pour le groupe de PO aussi.	avec toutes les équipes	demande et en fin de release
-----------------	-------------------------	-----------------------------	-------------------------	------------------------------

21.6. Planification à grande échelle

21.6.1. Plan de release multi-équipes

Chaque équipe possède son plan de sprint, et il y a un seul plan de release global. Il ne serait pas lisible au niveau des stories, il est donc élaboré avec des features.

Les techniques que nous avons présentées dans le chapitre 16 *Planifier la release* s'appliquent au tableau de features. La vitesse utilisée est la vitesse globale, basée sur les features finies par sprint, soit en les comptant, soit en points.

21.6.2. Feuille de route

Un plan de release montre le contenu d'une release. Sur un produit, on aimerait avoir une vue qui porte plus loin dans le temps. Une ligne chronologique, sur un horizon de 12 mois à 18 mois, qui montre les releases successives, avec leur nom et leur objectif, apporte une première réponse. Pour mieux appréhender la stratégie de développement du produit, on peut y ajouter les impacts et les équipes susceptibles d'y travailler.

21.7. Scrum à plusieurs équipes sur le terrain

21.7.1. Faire des scrums de scrums avec un réel besoin

Même si le scrum de scrums et plus largement le Scrum à grande échelle apparaissent séduisants, attention à ne pas se lancer sans un vrai besoin.

Un expert considérait il y a quelques années que c'était la dernière chose à essayer [Fowler, *LargeAgileProjects*] : un gros projet se décompose toujours en sous-projets et finalement on peut choisir de ne mettre en œuvre les méthodes agiles que sur ces sous-projets, avec une équipe agile standard.

Chef ... faudrait changer d'échelle !

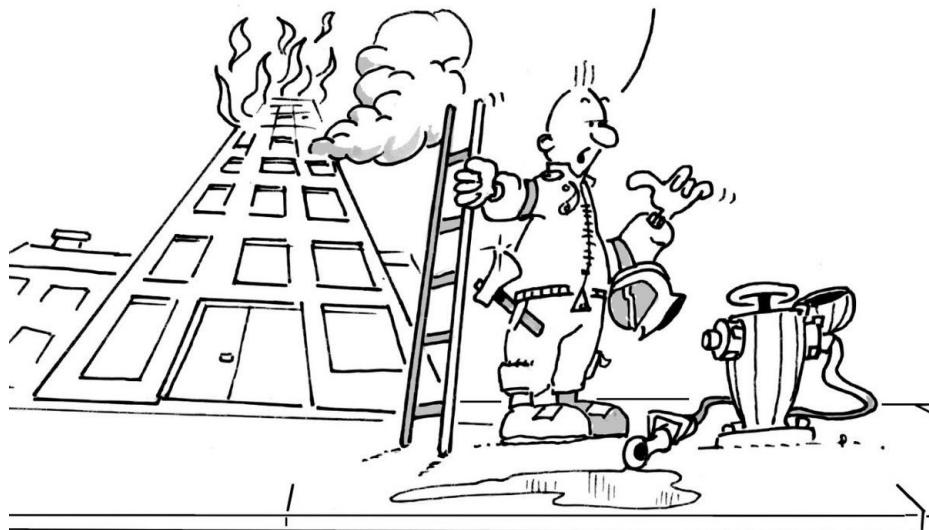


Fig. 21.9 Il faut un vrai besoin pour changer d'échelle

21.7.2. Mixer Scrum et Kanban

Si Scrum, avec son rythme donné par les sprints et les releases, s'applique bien pour suivre les équipes, un suivi de type Kanban est bien adapté pour le produit et ses features.

Le tableau Kanban de features est l'outil essentiel pour la direction. À l'autre extrémité, pour les remontées des utilisateurs, défauts et incidents, ce qu'on appelle le support, l'utilisation de Kanban est aussi adéquate.

21.7.3. Limiter la taille des backlogs

L'organisation en impacts, features et stories permet de limiter la taille des backlogs, en particulier des bacs d'affinage. Avec l'agilité, on passe dans un système de flux dont les backlogs sur plusieurs niveaux constituent les files d'attente.

L'organisation hiérarchique impact – feature – story – tâche permet de limiter leur taille, en se basant sur leur durée typique.

	Durée de réalisation
Impact	3 mois – 1 an
Feature	2 semaines – 2 mois
Story	1 jour – 1 semaine
Tâche	2 heures – 2 jours

21.7.4. L'expérience Spotify

En 2012, la publication d'une expérience multi-équipes significative [Kniberg, Spotify] a provoqué beaucoup d'intérêt. À tel point que certains ont voulu en faire un modèle pour l'agilité à grande échelle. Cependant, et comme le rappelle Kniberg, il ne s'agit pas de cela, mais simplement d'une expérience très intéressante, dans un contexte spécifique. On notera la mise en place d'équipes feature, l'importance donnée aux pratiques d'ingénierie et à la communication entre les équipes.

Bien commencer

La question à se poser

Comment passer à l'échelle sans perdre la puissance et la simplicité de Scrum ?

De mauvais signes

Le Scrum à grande échelle passe essentiellement par l'outillage informatique.

On utilise un framework d'agilité à grande échelle et c'est très lourd.

Pour apprendre

Participer au jeu que j'ai mis au point pour apprendre le Scrum à grande échelle, basé sur la construction d'un puzzle à plusieurs équipes

(<http://www.aubryconseil.com/pages/puzzle>).

Par quoi démarrer

Organiser les équipes « feature ».

Une lecture pour tous

Feature team [Larman, *Feature team*], un article traduit en français.

À retenir

Scrum n'est pas réservé aux petits projets. L'application à grande échelle est possible, mais doit rester dans l'esprit « agile ».

Tout le monde suit le même rythme des sprints et des releases.

Le point-clé est de constituer des équipes « feature », responsables de produire de la valeur. Travailler à plusieurs équipes nécessite d'ajuster le rôle des Product Owners, qui forment un groupe. Un seul tableau de features suffit, chaque équipe ayant son backlog de stories.

Les sprints se déroulent de façon presque habituelle pour les équipes. On ajoute une séance d'affinage global pour affecter les features et des mêlées de synchronisation entre les équipes et les PO.

Références :

- ☞ Henrik Kniberg et Anders Ivarsson, *Agilité à grande échelle chez Spotify*, VF, 2012.
http://www.fabrice-aimetti.fr/dotclear/public/traductions/SpotifyScaling_fr.pdf
- ☞ Martin Fowler, *LargeAgileProjects*, 2003.
<http://martinfowler.com/bliki/LargeAgileProjects.html>
- ☞ Craig Larman et Bas Vodde, *Feature Team Primer*, VF, 2010, <http://wiki.ayeba.fr/Equipe+Feature>.

Notes

[1] Celle du PMI, *Project Management Institute*, une référence en gestion de projet.

Transformer les organisations

La première fois que j'ai accompagné une grande entreprise dans la transition à Scrum, on m'avait bien recommandé de ne pas citer Scrum, ni d'utiliser son vocabulaire. Le backlog s'appelait le référentiel des exigences et les sprints étaient des itérations. Je ne parlais pas de mêlée mais de point quotidien. C'était avant que Scrum soit à la mode, et il ne fallait pas trop heurter les thuriféraires de la « méthodologie officielle ».

Maintenant, mes missions d'accompagnement à l'adoption de Scrum se font avec beaucoup plus de soutien et de communication. Elles ont aussi changé de nature : il s'agit de plus en plus souvent de transformer une organisation qui a commencé avec Scrum et veut aller plus loin.

J'ai rencontré de nombreuses situations qui ont déclenché le passage à Scrum, soit sur un projet pilote, soit sur un projet en pleine crise, ou encore à l'initiative de la direction pour tous les projets.

Adopter Scrum implique, pour une équipe, un changement dans la façon de travailler. Nous avons vu comment faire. Mais une équipe n'est jamais indépendante du reste de l'organisation et, *a fortiori* quand on élargit l'usage de Scrum à plusieurs équipes, cela a des impacts sur l'organisation.

C'est sur cette transformation « agile » d'une organisation que porte ce chapitre. Parfois elle est relativement facile, souvent elle est très difficile. Parfois elle est souhaitée, des fois elle est subie.

En introduction, je tiens à dire que je ne suis pas un spécialiste des transformations d'organisation en général. En revanche, je commence à avoir de l'expérience dans celles qui sont provoquées par Scrum, et c'est ce que je souhaite partager dans ce chapitre.

22.1. Pourquoi se transformer ?

22.1.1. Qu'appelle-t-on une transformation agile ?

Souvenons-nous de l'objectif de Scrum présenté au premier chapitre :

Scrum aide les gens à améliorer leur façon de travailler.

La mise en œuvre de Scrum avec une équipe, et même avec plusieurs équipes, peut commencer sans transformer l'organisation. On parle de transformation agile (avec Scrum) quand l'organisation prend conscience que, pour obtenir les bénéfices attendus, elle doit changer, changer de structure, voire changer de culture.

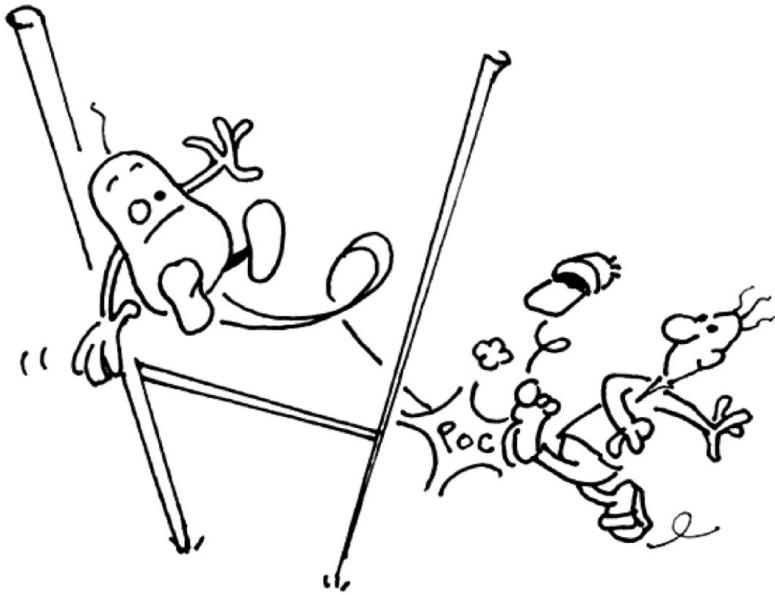


Fig. 22.1 Un ScrumMaster tente et réussit sa transformation agile

Revenons également à notre définition de l'agilité :

L'agilité est la capacité d'une organisation à fournir tôt et souvent des services ayant un véritable impact sur ses utilisateurs, tout en s'adaptant à temps aux changements dans son environnement.

Ce qui nous permet de tenter d'exprimer ce qu'est une transformation agile :

La transformation agile correspond à la démarche que met en œuvre une organisation pour acquérir cette capacité.

22.1.2. Origines de la transformation

Certaines organisations ont la volonté de se transformer dès le départ, lorsqu'elles décident de passer à Scrum.

Pour d'autres, cela vient, parfois par surprise, pendant l'expérimentation, quand l'équipe fait remonter des obstacles qu'elle ne peut pas éliminer, faute d'avoir le pouvoir suffisant pour cela.

Souvent, Scrum est considéré par l'organisation comme une simple méthode de développement ; au début, il n'est pas question de se transformer.

Mais comme Scrum est terriblement efficace pour faire émerger et rendre visibles les obstacles, on arrive vite à la situation où l'organisation va devoir se poser la question de se transformer. Elle peut alors décider de la refuser ou de la repousser, c'est toujours un choix possible qui forcément limitera les bénéfices obtenus.

Dans la plupart des cas, c'est l'apparition d'obstacles d'organisation qui va déclencher la transformation.

22.1.3. Obstacles d'organisation

Les obstacles identifiés par une équipe, qui la freinent ou la bloquent dans le développement du produit, sont variés ; ils proviennent :

- des pratiques de Scrum ;
- des pratiques complémentaires à Scrum, notamment les pratiques d'ingénierie ;
- de la résistance de l'organisation au changement ;
- de son type de gouvernance.

Pour les deux premiers, l'équipe peut, le plus souvent, trouver des solutions en son sein. Pour les deux derniers, elle doit escalader vers le management, ou vers d'autres parties prenantes.

Un premier pas de l'organisation dans sa transformation est la collecte des obstacles qui la concernent, pour définir les plus prioritaires et les afficher, en évidence pour tout le monde.

Nous avons conseillé dès le premier chapitre de « faire du Shu », c'est-à-dire de démarrer Scrum en suivant les pratiques et, un peu plus loin dans le livre, de « contextualiser Scrum ». Cela n'est pas contradictoire. La friction ainsi créée permet de faire remonter rapidement les obstacles d'organisation, dans deux situations typiques :

- quand une équipe n'arrive pas à « faire du Shu » à cause du contexte ;
- si elle souhaite passer du « Shu » au « Ha ».

Leur résolution demande une implication du management ou d'autres entités. Leur élimination définitive passe bien souvent par une transformation de l'organisation.

22.1.4. Un problème de culture

Car, si on arrive à régler des obstacles d'organisation, l'analyse de leur cause profonde fait souvent ressortir qu'ils sont dus à des problèmes de culture. Il y a des chances qu'ils reviennent si on ne change pas la culture de l'organisation.

La culture d'entreprise est une notion bien souvent négligée par le management. Privilégiant les objectifs financiers à court terme, les entreprises ne cherchent pas à devenir des collectifs organiques, c'est-à-dire des regroupements de personnes complémentaires, organisés autour d'un sentiment commun d'appartenance, comme peut l'être une véritable équipe Scrum.

La culture d'entreprise est un paramètre essentiel de la transformation à l'agilité. L'introduction de Scrum, avec son nouveau langage et ses nouveaux rites, provoque inéluctablement un impact sur sa culture.

Ce sont les entreprises qui se préoccupent de la culture d'entreprise qui seront les mieux armées pour réussir la transformation. Celles qui considèrent Scrum uniquement comme levier d'augmentation de la productivité et ne tiennent pas compte du volet culturel ont toutes les chances d'échouer.

Finalement, beaucoup considèrent que le développement agile est plus une culture qu'un processus [Patton, *Agile Culture*].

Le mouvement agile possède des valeurs et des principes qui constituent sa culture ; il faut en tenir compte dans la transformation à l'agilité.

Effectivement, il s'avère que la transformation est difficile si la culture de l'entreprise n'est pas en résonance avec celle poussée par Scrum, qui est celle de collaboration. Elle risque, par exemple, d'être longue et délicate pour une organisation plus dans la culture du contrôle, que pour une davantage focalisée sur la culture de la compétence individuelle [Sahota, *Transformation*].

22.2. Comment se transformer ?

22.2.1. Avec qui ?

Une approche congruente serait que la transformation agile se fasse aussi en appliquant Scrum. Considérer la transformation comme un projet ou un programme et le mettre entre les mains des spécialistes des processus n'est assurément pas une bonne idée.

Pourtant, les grandes organisations possèdent des « agents du changement », et font régulièrement des « réorganisations ». Mais, le changement ainsi imposé à leurs employés ne donne rien de bon la plupart du temps.

Dans une transformation agile, il est préférable d'entraîner toutes les personnes qui sont sur le terrain, et de procéder par invitation pour les actions concrètes.

Dans les grandes entreprises, une transformation sera d'abord circonscrite à un service ou un département. Impliquer une cinquantaine de personnes est bien suffisant dans un premier temps. La participation du grand manager du département ou de l'organisation est indispensable.

22.2.2. Définir l'objectif de la transformation

Qu'est-ce qui pousse l'organisation à adopter à Scrum ? Quel est son objectif en introduisant l'agilité ?

Il faut avoir une idée claire de la raison du changement : passer à Scrum parce que c'est à la mode n'est évidemment pas une justification suffisante.

Baser la transformation sur des attentes floues expose à la déception.

Si vous avez établi une liste des obstacles, elle servira de point de départ. On peut aussi organiser une rétrospective sur les derniers projets réalisés. Les participants à ces projets sont conviés pour une séance de réflexion collective dont le but est d'identifier cette liste de points à améliorer.

Une autre possibilité est de faire appel à un expert externe, un « consultant agile », pour effectuer des interviews des gens de l'organisation.

Exemples de problèmes pouvant être remontés :

- On fait trop de projets à la fois.
- Il y a une pression terrible en fin de projet, on travaille toujours dans l'urgence.
- La qualité est déplorable parce que les tests sont négligés, les clients ne sont pas satisfaits...

Attention, après cela, il s'agit ensuite de converger vers un seul objectif clair. La mise en évidence du problème principal, celui que la transformation à Scrum cherche à résoudre, permet de se focaliser vers cet objectif et de motiver les personnes impliquées dans le changement.

Les ateliers et outils que nous avons présentés précédemment peuvent s'avérer utiles pour aboutir à une vision de la transformation (par exemple, *impact mapping* pour la stratégie suivie).

Comme un objectif est toujours difficile à déterminer dans ce contexte, un modèle [Shore, *Agile fluency*] peut nous y aider. Nous y reviendrons un peu plus loin.

22.2.3. Façons de transformer

La transformation est de la gestion du changement. Le cycle typique d'une conduite du changement « classique » avec un projet pilote dans une grande organisation montrerait les activités suivantes :



Fig. 22.2 Les activités typiques de conduite du changement

On pourrait s'appuyer sur cette approche pour lancer notre transformation agile. Cependant, cette approche trop linéaire n'est pas la meilleure façon de s'attaquer à la complexité d'une transformation agile. De plus, toutes les organisations étant différentes, il n'est pas question de définir la « bonne façon » de faire la transformation.

Dans une volonté de congruence, une transformation devrait s'appuyer sur les valeurs et les principes de Scrum.

Réflexions sur l'évaluation de l'agilité et sa capitalisation

On pourrait se dire qu'une entreprise qui effectue une transformation se fait une idée des progrès réalisés en effectuant la collecte de mesures pertinentes et en évaluant le niveau d'agilité atteint.

Mais, comme nous l'avons vu dans le chapitre 18 *Améliorer la visibilité avec les indicateurs*, il n'y a pas d'indicateurs « processus » pour Scrum. Le risque est énorme que cela se fasse sur les pratiques, alors que ce qui compte, ce sont les résultats (la « valeur métier » fournie).

Dans la conduite du changement classique, la capitalisation est aussi au cœur de la transformation : en se basant sur les succès et les échecs des projets pilotes, elle cherche à influer sur les processus de l'entreprise. Le canal habituel de diffusion reste l'écrit.

Avec l'agilité et Scrum, on priviliege plutôt les conversations et la transmission orale, même pour la transformation.

Cependant, la communication sous forme écrite n'est pas bannie pour autant. Elle peut être utile pour partager :

- la contextualisation de Scrum pour différentes équipes,
- la liste des pratiques qui ont été appliquées,
- celle des obstacles rencontrés et l'analyse de leur cause,
- les définitions de fini et de prêt, ainsi que les storyotypes.

En cas de projet pilote avec Scrum, les résultats sont communiqués largement dans l'entreprise, pour commencer à créer une communauté et déclencher de nouvelles vocations dans les équipes. Les participants au projet pilote sont le vecteur idéal pour diffuser la connaissance dans l'organisation.

22.2.4. Principes de transformation agile

Les six principes (EPATER) qui font la particularité de Scrum, présentés dans le premier chapitre, sont repris comme fondamentaux d'une transformation agile.

- **Empirisme**

Une transformation agile est complexe, on ne sait pas à l'avance jusqu'où on va précisément aller. Une approche empirique s'impose, avec des boucles de feedback impliquant les parties prenantes de la transformation. Il n'est pas question de décrire le processus cible à l'avance.

L'objectif est soutenu par le manager, qui est le sponsor de l'initiative. Il sert de vision et les avancées de la transformation sont « inspectées et adaptées » par ceux à qui elle est destinée.

- **Priorité**

Comme nous l'avons vu pour la rétrospective de sprint, il y a toujours beaucoup plus à améliorer que ce que l'équipe peut raisonnablement faire. C'est la même chose pour la transformation, c'est pourquoi les actions sont priorisées.

- **Auto-organisation**

Les actions de transformation demandent du temps ou de l'argent pour être réalisées. Le sponsor de la transformation définit le cadre et le budget, ce sont les parties prenantes qui définissent la meilleure façon de procéder.

L'auto-organisation facilite la progression de chacun auprès de sa propre culture. Elle pousse les gens à prendre des initiatives, à se lâcher, à favoriser les idées disruptives.

*On a besoin d'un
nouveau poste d'usinage. Avec une tireuse à bière alors...*

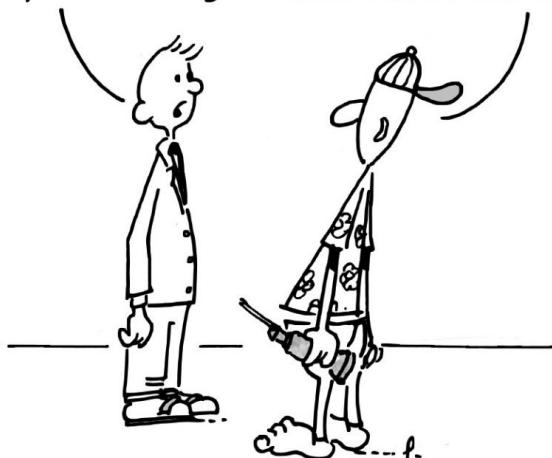


Fig. 22.3 Une idée disruptive

- **Transparence**

Les décisions sont prises collectivement, ce qui renforce l'engagement collectif. Des techniques de vote prenant en compte les choix secondaires sont préférables.

Le suivi des actions décidées est affiché de manière à être visible de toute l'organisation. Une présentation de type tableau Kanban est tout à fait adaptée.

- **Émergence**

Les idées de transformation ne viennent pas d'en haut, ou d'un groupe de spécialistes qui y a réfléchi. Ce sont les gens qui les font émerger eux-mêmes, à partir de leur vécu et de leur ressenti. Tout le monde n'est pas obligé de participer et d'avoir des idées. Les personnes qui viennent et contribuent sont les bonnes.

Une transformation agile favorise la notion d'invitation. L'idée est de renforcer l'engagement, qui sera d'autant plus fort qu'il sera librement consenti.

L'émergence tient également au fait qu'il n'y a pas de programme préétabli qui donnerait déjà des axes de réflexion. À partir de l'objectif de la transformation, ce sont les participants qui proposent les sujets qui leur tiennent à cœur.

- **Rythme**

Le rythme idéal pour franchir un palier est celui de la release. La rétrospective de fin de release est le moment propice pour inviter les gens à participer.

22.3. Trois obstacles de nos organisations

Scrum véhicule une vision qui est, la plupart du temps, radicalement différente sur la façon de s'organiser et de gérer les ressources humaines. Voici quelques exemples de pratiques anciennes remises en question par une transformation agile.

22.3.1. MOA et MOE ne sont pas agiles

Dans des grandes organisations françaises, il existe encore une division entre la maîtrise d'ouvrage (MOA), qui représente des utilisateurs internes à l'entreprise, et la maîtrise d'œuvre (MOE), dans laquelle on trouve la direction des systèmes d'information (DSI).

Une organisation MOA/MOE tend à créer une séparation très nette entre deux entités. Cela contribue à :

- baser la communication sur des documents au détriment de la communication orale ;
- écrire des spécifications détaillées dès le début du projet ;
- passer des tests d'acceptation tardivement.

Scrum propose des pratiques radicalement différentes.

Aussi plutôt que de se demander si c'est la MOA ou la MOE qui doit jouer le rôle de Product Owner, une transformation pousse à se débarrasser de ces notions et de ce qu'elles impliquent. Il convient de faire tomber les murs et de créer une seule équipe qui contribue au même objectif (qui n'est pas d'écrire de la documentation). Le Product Owner fait partie de cette équipe avec ceux qui analysent, conçoivent, développent, testent, rédigent, etc.

Une responsabilité habituelle de la MOA est le déroulement des tests de recette. Avec Scrum, il n'y a pas deux équipes, une de développement et l'autre de test, mais une seule équipe accueillant tous les participants au projet, y compris les testeurs, présents dans l'espace de travail.

22.3.2. Les évaluations individuelles sont contre-productives

Esther Derby (dans l'article *Performance without Appraisal*^[1]) critique l'évaluation individuelle au mérite. Elle cite notamment Deming, bien connu pour ses travaux sur la qualité (et pour sa roue^[2]) :

« L'idée de la récompense au mérite est séduisante. La sonorité du slogan est attractive : payez pour ce que vous obtenez, obtenez ce que vous payez, motivez les gens à faire de leur mieux, pour leur propre bien. L'effet est exactement le contraire de ce que les mots promettent. Chacun se met en avant, ou essaie, uniquement pour sa récompense, de tirer la couverture à lui. L'organisation est la grande perdante. Cela récompense le mérite des gens qui profitent du système. Cela ne récompense pas les efforts faits pour améliorer le système »

W. Edwards Deming, Out of the Crisis.

Scrum pousse à supprimer les entretiens annuels d'évaluation couramment pratiqués dans les entreprises et fournit des réponses intéressantes aux questions qui en découlent :

- comment déterminer le salaire de chacun,
- comment donner une promotion à une personne ou au contraire la « virer »,
- comment les gens sauront qu'ils doivent s'améliorer.

Scrum, par ses mécanismes de régulation et la transparence apportée, contribue à privilégier une approche collective. La transformation agile remet en question le principe des évaluations individuelles.

22.3.3. Les contrats sont trop rigides

Le malentendu entre contrat dit au forfait et agilité vient de l'idée que le périmètre fonctionnel semble être fixé à l'avance par le client. Mais c'est un leurre : sauf pour de petits projets où on peut se passer de feedback ou dans les cas particuliers de spécification formelle, l'expérience montre que c'est impossible. Dans la réalité, le périmètre n'est pas strictement défini au début du projet, et évolue toujours.

Avec une méthode agile, partant de ce constat et du postulat de base qui est l'acceptation du changement, on considère que le périmètre est la variable d'ajustement.

Il existe de nombreuses façons de rendre les contrats plus agiles. Pour en citer quelques-unes :

- **Établir un contrat à engagement de moyens** (et pas de résultat).
- **Décomposer l'appel d'offres en deux parties** : une première pour évaluer la capacité du fournisseur (ou des fournisseurs) et une deuxième avec un engagement sur sa capacité.
- **Faire un contrat avec une enveloppe fixée et un périmètre évalué par morceaux**, de façon similaire à ce qui se pratique pour les TMA (notion d'unité d'œuvre pour la tierce maintenance applicative).
- **Faire un contrat de façon habituelle, mais en introduisant de nouvelles clauses agiles.**

Dans tous les cas, la façon de faire les appels d'offres devrait être adaptée pour introduire une façon de procéder plus agile. Le backlog est un élément essentiel de la démarche agile. Il peut, selon les caractéristiques du projet, être annexé à l'appel d'offres ou réalisé en commun entre client et fournisseur.

Le chapitre 16 *Planifier la release* donne de nombreuses pistes pour faire des appels d'offres avec une gestion de projet plus agile.

Impact de la clause « changement gratuit » dans un contrat

Cette clause offre au client la possibilité de changer d'avis sans que cela remette en cause le contrat, en restant à prix constant. Le principe est le suivant : à chaque fin de sprint, le client peut changer d'avis sur le contenu du backlog (dans sa partie bac d'affinage), à condition que ce qu'il ajoute soit contrebalancé par ce qu'il retire. Cela suppose que les éléments du backlog soient interchangeables, avec une monnaie d'échange comme les points de story. Pour cela, le client et le fournisseur doivent avoir acquis une confiance mutuelle et le Product Owner doit jouer vraiment son rôle en participant activement au projet.

L'intérêt pour le client est qu'il peut avoir plus de valeur métier pour un prix équivalent. En fait, ce type de changement est déjà pratiqué dans la plupart des projets au forfait au cours de négociations entre client et

fournisseur, mais souvent en cachette puisque c'est contraire au principe du périmètre fixé à l'avance, qui est la base des contrats classiques.

En conclusion sur ce sujet qui est l'objet de nombreuses discussions, la transformation agile passe par l'adaptation des contrats. De très nombreux projets sont réalisés avec Scrum dans un cadre contractuel et, heureusement, si ça leur ajoute quelques contraintes, ça ne les empêche pas de réussir.

22.4. Des outils pour la transformation

22.4.1. Le modèle Agile Fluency

Ce modèle « Agile Fluency » [Shore, *Fluency*] utilise la perception de la valeur « métier » produite par une équipe pour situer sa « maîtrise de l'agilité », en termes d'étoiles. Il est intéressant pour la transformation, car il permet d'aligner les efforts de transformation avec les attentes. Appliqué au départ pour une équipe, il montre qu'un changement d'organisation devient nécessaire si l'objectif est ambitieux.

A Team's Path Through Agile Fluency

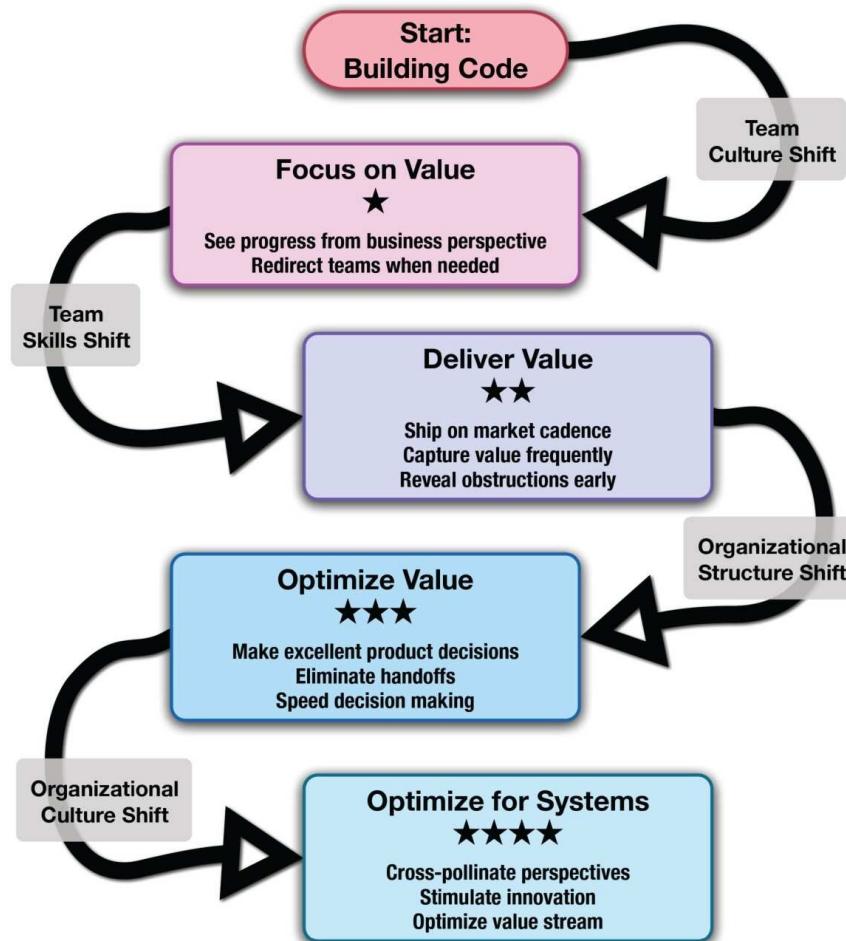


Fig. 22.4 Le modèle Agile Fluency (source : © 2012 James Shore et Diana Larsen).

Chaque étoile correspond à un palier typique, pour lequel on peut donner une idée des résultats obtenus.

Nous allons utiliser ce chemin vers l'agilité, avec ses paliers, pour lister les pratiques et les outils à mettre en place dans une tactique de transformation basée sur Scrum et tout ce que nous y avons ajouté dans les différents chapitres. Nous évoquerons de nouveaux outils, qui seront des pistes pour la transformation.

22.4.2. Une étoile ★

Objectif	Focus sur la valeur métier.
Palier	L'équipe a acquis la culture Scrum.
Résultat	Elle produit un incrément de sprint et les parties prenantes se rendent compte, lors de la revue, des progrès dans la création de valeur.
Pratiques à mettre en place	Toutes les pratiques de base de Scrum : le sprint, les rôles, le backlog et le rituel du sprint (planification, mêlée, revue et rétrospective).
Formation pour y parvenir	Une formation à Scrum pour toute l'équipe.
Outil d'amélioration pour agir sur la transformation	La rétrospective.

22.4.3. Deux étoiles ★★

Objectif	Fourniture régulière de valeur.
Palier	L'équipe a acquis des compétences d'ingénierie complémentaires.
Résultat	Elle produit régulièrement, ce qui permet de livrer de la valeur à une cadence souhaitée par le « métier ».
Pratiques à mettre en place	<ul style="list-style-type: none"> – Dans le cadre Scrum, l'accent est mis sur l'affinage, la définition de fini et la définition de prêt. – Les pratiques XP de développement sont approfondies. – Des pratiques complémentaires comme DevOps sont ajoutées.
	<ul style="list-style-type: none"> – Aux pratiques de développement que l'équipe ne maîtrise pas.

Formation pour y parvenir	– Du coaching peut aider l'équipe dans les pratiques Scrum plus poussées.
Outil d'amélioration pour agir sur la transformation	En plus de la rétrospective, l'intraspective, c'est-à-dire que l'équipe n'attend pas la fin du sprint pour éliminer les obstacles et analyser les raisons profondes de leur apparition.

22.4.4. Trois étoiles ★★★

Objectif	Optimisation de la valeur fournie.
Palier	L'organisation a transformé sa structure.
Résultat	Elle prend les bonnes décisions sur le produit, ce qui permet de maximiser l'impact en minimisant les efforts (en anglais : « <i>maximise outcome, minimise output</i> »).
Pratiques à mettre en place	<ul style="list-style-type: none"> – Les pratiques de découverte du produit que nous avons présentées (impact mapping, story mapping, lean startup). – Des pratiques d'organisation vues également (la mise en place d'équipes feature, la visualisation à haut niveau avec un kanban de features). – De nouvelles pratiques de management (les outils du Management 3.0 [Appelo, <i>Management 3.0 workout</i>], les communautés de pratiques).
Formation pour y parvenir	<ul style="list-style-type: none"> – Pour les acteurs de la transformation, une formation différente qui met les participants en immersion, comme le raid Agile. – Pour toutes les parties prenantes, une sensibilisation d'une journée à Scrum et l'agilité est nécessaire.
Outil d'amélioration pour agir sur la transformation	<i>Open Agile Adoption</i> [Mezick, OAA]

Open Agile Adoption

OAA propose un cycle régulier qui colle bien au rythme des releases. Le « rite de passage » est un événement donnant une grande place à la culture d'organisation, qui aide les gens à appréhender des transitions complexes.

Il se déroule selon le principe du forum ouvert^[3].

Dans une journée style forum ouvert, les sujets de proposition sont « achetés » par les participants intéressés.

En fin de séance, les réflexions de chaque groupe apportent des idées concrètes pour la transformation. Il convient de prioriser pour ne conserver que quelques actions à mener pendant la release.

Pour en savoir plus, on lira le récit d'une journée d'*Open Agile Adoption* [Pernot, *journée OOA*].

22.4.5. Quatre étoiles ★★★★

Objectif	Optimisation du système complexe que représente l'organisation.
Palier	L'organisation a transformé sa culture.
Résultat	<ul style="list-style-type: none">– Elle optimise la création de valeur avec plusieurs équipes qui collaborent avec une pollinisation réciproque.– La notion de projet est supprimée, pour aller vers un flux de services déployés plus fréquemment.
Pratiques à mettre en place	<ul style="list-style-type: none">– Renforcer les pratiques mises en place en favorisant l'innovation.– Mettre en place des pratiques d'organisation innovantes, comme la sociocratie [Sarrazin, <i>Rupture douce 1</i>],– Élargir la transformation à la notion d'entreprise libérée [Sarrazin, <i>Rupture douce 3</i>].

Bien sûr, la transformation n'est jamais terminée, c'est un chemin qu'on poursuit.

Bien commencer

La question à se poser Quel est l'objectif de la transformation agile ?

L'ancien processus est toujours là, les anciennes habitudes reviennent dès la première difficulté.

De mauvais signes La culture du contrôle reste solidement implantée, avec des indicateurs de micromanagement toujours demandés par le management.

Par quoi démarrer

Une lecture pour tous

Faire des rétrospectives de sprint et des rétrospectives plus poussées en fin de release, de type open space.

Les trois saisons de *Rupture Douce* [Sarrazin, *Rupture douce*].

À retenir

Si lancer en équipe avec Scrum est relativement facile, la transformation d'une organisation est autrement plus délicate.

Pour certaines organisations, on pourrait parler de rupture ou de mutation, comme on parle de mutation industrielle : un changement économique et social brusque et spectaculaire, qui entraîne une modification profonde.

Les techniques habituelles de conduite du changement ne sont pas adaptées. Une orientation sur les résultats attendus est préférable, permettant de définir une stratégie de transformation alignée.

L'idée forte est de baser la transformation sur les valeurs de Scrum (engagement, focalisation et courage notamment) et de s'appuyer sur les principes pour agir sur la culture.

Références :

- ☞ Jurgen Appelo, *Management 3.0 Workout*, <http://www.management30.com/>.
- ☞ Dan Mezick, *Open Agile Adoption*, Web, <http://www.infoq.com/fr/articles/open-agile-adoption-2>.
- ☞ Dan Mezick, *The culture game*, <http://agilelib.net/?100013>.
- ☞ Michaël Sahota, *Agile Adoption and Transformation*, VF, 2013, <http://www.agiliste.fr/livres/guide-transformation-agile/>.
- ☞ Jeff Patton, *Agile Development is more Culture than Process*, Web, 2009, http://www.agileproductdesign.com/blog/agile_is_culture_not_process.html.
- ☞ Pablo Pernot, *Open Agile Adoption en une journée*, Web, <http://www.areyouagile.com/2014/04/histoires-dopen-agile-adoption/>.
- ☞ Laurent Sarrazin et d'autres, *Rupture Douce*, les trois saisons, <http://laurentsarrazin78.wix.com/rupture-douce>.

Notes

- [1] <http://www.scrumalliance.org/articles/50-performance-without-appraisal>.
- [2] http://fr.wikipedia.org/wiki/Roue_de_Deming.
- [3] https://fr.wikipedia.org/wiki/M%C3%A9thodologie_Forum_Ouvert

Quiz

Ce quiz a été testé et mis au point avec les participants au premier panier-repas agile Montpellier-Toulouse qui s'est déroulé le 15 juillet 2011 à Bages (Aude). Il a depuis été joué des dizaines de fois et amélioré grâce à ce feedback. Il est aussi utilisé lors de la randonnée à Sainte-Croix de Caderle (Gard), à chaque raid Agile^[1] en Cévennes et aide à préparer la certification CSM (Châtaigne, Saucisson et Méthode agile).

Les questions et leurs réponses sont présentées et commentées dans mon blog *Scrum, Agilité et Rock'n'roll* : aubryconseil.com.

Pour chaque question, vous avez le choix entre quatre réponses possibles. À vous de choisir celle qui vous semble la plus appropriée dans la situation décrite.

Scrum et l'agilité

Ce n'est pas une valeur compatible avec Scrum :

- A. La certitude.
- B. Le respect.
- C. L'humour.
- D. Le courage.

Le cycle des sprints

Pour définir la durée des sprints, comment fait l'équipe Scrum ?

- A. Elle applique la formule $d = (n^2 - 1)/a$, où n est la taille de l'équipe et a l'âge moyen.
- B. Elle lance un dé à quatre faces.
- C. Elle élabore d'abord la définition de fini.
- D. Elle demande au ScrumMaster.

Les gens de Scrum

Le sprint de trois semaines a commencé vendredi, avec une équipe de dix personnes. Le lundi lors de la mêlée du matin, on apprend qu'un développeur s'est cassé le bras droit au ski, il est plâtré pour une semaine :

- A. Le ScrumMaster interdit le ski à toute l'équipe jusqu'à la fin de la release.
- B. L'équipe revoit son engagement sur le sprint.
- C. On lui trouve un remplaçant pendant qu'il est plâtré.
- D. On verra ce que ça donne et en attendant on lui achète une souris pour gaucher.

Le rôle du Product Owner

Une partie prenante clé du projet tient beaucoup à une fonction mais ne sait pas bien de quelle façon elle pourrait être proposée aux futurs utilisateurs du produit. Vous êtes Product Owner, que faire ?

- A. Attendre que cette partie prenante mette clairement ce qu'elle veut dans le bac à sable.
- B. Définir une story simple avec ce que vous avez compris et la mettre prioritaire.
- C. Mettre sa demande dans le bac à glace en attendant que ça se décante.
- D. Lui demander d'écrire la spécification détaillée.

Le rôle du ScrumMaster

Une personne dans l'équipe perturbe les autres. Vous êtes ScrumMaster, comment réagissez-vous ?

- A. Je ne fais rien, c'est ça l'auto-organisation.
- B. Je la vire, la majorité est d'accord.
- C. Je fais un rapport à la direction du personnel.
- D. Je l'invite à prendre une bière dans une discussion en tête-à-tête pour lui proposer d'être le ScrumMaster du prochain sprint.

Structurer le backlog

On parle d'epic à propos du backlog, qu'est-ce que cela désigne ?

- A. #epic fail. C'est une story qui n'est pas finie à la fin du sprint.
- B. C'est une story prête.
- C. C'est une story trop grosse.
- D. C'est une story proposée par une partie prenante, en référence à la fable de la poule et du porc-épic.

Affiner le backlog

Une story prête, actuellement dans le bac de départ, est finalement jugée inutile par le Product Owner.

- A. On la supprime.
- B. On la fait quand même car elle était prévue dans le prochain sprint.
- C. On la garde en fin du bac d'affinage.
- D. On la conserve dans le bac à glace, au cas où le PO changerait encore d'avis.

Définition de fini

La définition de fini dit que les règles de codage doivent être vérifiées à chaque sprint. Une règle est violée à la fin d'un sprint. Que faire ?

- A. Noter le problème sur un Post-it® rouge.
- B. Prolonger le sprint le temps de rectifier le code.
- C. Changer la définition de fini.
- D. Ajouter une story de remboursement de dette technique dans le backlog.

Planifier le sprint

Le Product Owner veut ajouter une story qui n'est pas dans le bac de départ et que l'équipe estime à risque, difficile à finir dans ce sprint. Vous faites partie de l'équipe, quelle est votre réaction ?

- A. Vous refusez la story.
- B. Vous la prenez, le PO insiste en disant qu'elle a beaucoup de valeur.
- C. Vous essayez de décomposer la story pour ne prendre que le morceau bien compris.
- D. Vous demandez une prime.

Mêlée quotidienne

Dans une équipe de neuf personnes, la mêlée quotidienne est à 9 h 30. À l'heure prévue, deux membres de l'équipe ne sont pas là.

- A. On commence normalement.
- B. C'est le quart d'heure toulousain, le ScrumMaster raconte une blague en attendant.
- C. On repousse la réunion à 10 heures.
- D. On annule la mêlée aujourd'hui, on verra demain.

Revue de sprint

Pendant la revue, une partie prenante trouve un défaut dans l'interface utilisateur d'une story qui est présentée. Que faire ?

- A. Considérer la story comme non finie et la continuer dans le sprint suivant.
- B. Corriger le défaut en vitesse.
- C. Ajouter une entrée dans le bac à sable pour corriger ce défaut.
- D. Lui demander d'envoyer un message au PO.

Rétrospective

Vous êtes ScrumMaster. Lors de la rétrospective, Nico dit que c'est la faute de Lolo si la story S1 n'a pas été finie pendant le sprint.

- A. Vous donnez un carton jaune à Nico.
- B. Vous donnez un carton jaune à Lolo.
- C. Vous dessinez un diagramme en arête de poisson pour identifier les raisons du problème sur S1.
- D. Vous organisez un duel aux fléchettes.

Contextualiser Scrum

Une équipe répartie entre la France, l'Inde et la Roumanie fait la mêlée quotidienne par vidéoconférence. Quelle règle Scrum conservez-vous en priorité ?

- A. L'équipe doit rester debout.

- B. Les poules n'ont pas le droit de parler.
- C. Tout le monde répond aux trois questions.
- D. La mêlée ne dépasse pas un quart d'heure.

Découvrir le produit

Les prévisions évoluent, suite à des hypothèses vérifiées sur les impacts. Votre produit qui était prévu au départ pour 100 utilisateurs, devrait finalement en supporter 10 000. Que faites-vous ?

- A. Vous ajoutez une *user story* « Passer à 10 000 utilisateurs » dans le backlog.
- B. Vous revoyez la définition de fini.
- C. Vous ajoutez une story technique pour mesurer l'impact.
- D. Rien, c'est le problème de l'infrastructure.

Raconter la story

Une story présentée en démonstration fonctionne comme prévu, ses quatre tests passent, mais on y découvre un nouveau cas de test auquel on n'avait pas pensé. Que faire ?

- A. Elle est considérée finie à 80 %.
- B. Elle n'est pas considérée comme finie.
- C. On modifie le code vite fait.
- D. Elle est considérée comme finie et on ajoute une entrée dans le backlog pour le cas trouvé.

Planifier la release

Pour réaliser la story « Tableau de bord », il faut que le composant qui envoie les données fonctionne. Il doit être développé par une autre équipe. Vous êtes en train de faire l'affinage du backlog, à quelques jours du démarrage du prochain sprint.

- A. La story est planifiée dans le prochain sprint.
- B. La story ne peut pas être planifiée tant que le composant n'est pas fini.
- C. La story est planifiée dans le sprint après le suivant et on prévient l'autre équipe.
- D. L'équipe développe elle-même le composant.

Appliquer Kanban à Scrum

La limite dans la colonne « Test » de votre tableau de features est de 3 et c'est rempli. Vous êtes développeur en train de finir le code d'une feature, que faites-vous ?

- A. Vous ajoutez votre feature dans la colonne et vous faites + 1 sur la limite.
- B. Vous ralentissez votre développement.
- C. Vous aidez les personnes qui testent les features.
- D. Vous demandez au Product Owner.

Ajouter des pratiques de développement XP

L'application à développer s'appuie sur du code existant dont on sait que la qualité n'est pas exceptionnelle. Que faire ?

- A. Ajouter un test quand un bug est trouvé qui porte sur ce code existant.
- B. Améliorer ce code existant en priorité.
- C. Surtout ne pas toucher à ce code.
- D. Ajouter des tests pour couvrir tout le code existant.

Améliorer la visibilité avec des indicateurs

Au quatrième sprint, le burndown chart de release remonte. Vous êtes ScrumMaster, que faire ?

- A. Vous recomptez les points.
- B. Vous dites au PO que s'il continue à ajouter des stories, ça va coincer.
- C. Vous le présentez en le renommant burnup.
- D. Vous passez un savon à l'équipe.

Tirer profit des outils

Vous êtes en train de vérifier les conditions d'acceptation d'une story et une condition est en échec. Comment le noter sur le tableau de l'équipe ?

- A. Vous remettez la tâche de codage dans la colonne en cours.
- B. Vous ajoutez un Post-it® pour la tâche de correction dans la colonne à faire.
- C. Vous ajoutez un bug dans le bac à sable.
- D. Vous ajoutez un nouvel obstacle dans la liste.

Développer un produit à plusieurs équipes

Trois équipes Scrum participent au développement d'un seul produit, chacune s'occupant d'un ensemble de features. Comment gérer la définition de fini ?

- A. Une seule définition de fini pour tout le monde.
- B. Pas besoin, on en parle au scrum de scrums.
- C. Chaque équipe a la sienne et la communique aux autres.
- D. C'est le Product Owner qui décide.

Transformer les organisations

Une transformation agile se fait, de préférence, en :

- A. Tapant entre les barres après un essai de Scrum.
- B. Appliquant à l'organisation un framework d'agilité à grande échelle.
- C. Organisant un open space où tout le monde est invité.
- D. Formant les ScrumMasters aux bonnes pratiques.

Notes

[1] <http://raidagile.fr/>

Glossaire

Affinage du backlog : activité qui consiste à entretenir le backlog pour augmenter les chances de succès des futurs sprints.

Agilité : capacité d'une organisation à fournir tôt et fréquemment des services impactant ses utilisateurs, tout en s'adaptant à temps aux changements dans son environnement.

Bac à glace : dépôt des stories hors du périmètre de la release courante, qui y sont gelées jusqu'à la release suivante.

Bac à sable : dépôt non ordonné des idées et propositions de tout le monde pour le produit.

Bac d'affinage : partie du backlog sur laquelle s'effectue l'affinage.

Bac de départ : dépôt des stories prêtes pour un sprint.

Bac d'arrivée : dépôt des stories finies.

Backlog : dépôt de collecte et de partage des choses à faire, ou stories, qui apportent de la valeur.

Bug : imperfection qui enlève de la valeur au produit.

Burndown chart : graphique montrant la tendance du reste à faire.

Burnup chart : graphique montrant la tendance de ce qui est fini.

Capacité : utilisée pour la planification de release, prévision de ce que l'équipe est capable de faire pendant un sprint.

Condition d'acceptation : description d'une condition objective que l'équipe utilise pour déterminer si une story est acceptable. Pour qu'elle soit finie, il faut aussi vérifier sa définition de fini.

Définition de prêt : liste de vérifications pour s'assurer que le bac de départ contient suffisamment de stories pour le prochain sprint ; s'applique aussi à une story et dans ce cas permet de statuer sur sa possibilité de rentrer dans un sprint.

Définition de fini : liste de vérifications pour s'assurer de l'état de l'incrément à la fin du sprint ; s'applique aussi à une story et à une feature et dans ce cas permet de statuer sur leur finition.

Dette technique : code ou documentation de mauvaise qualité entraînant de la perte de temps (l'intérêt de la dette).

Développeur : membre de l'équipe Scrum.

Epic ou story épique : story dont l'équipe considère qu'elle est trop grosse pour rentrer dans un sprint en l'état.

Équipe auto-organisée : équipe qui possède le pouvoir de décider comment faire le travail.

Équipe Scrum : groupe de gens auto-organisé et pluridisciplinaire qui contribue à produire un incrément de produit à chaque sprint.

Équipe pluridisciplinaire : équipe qui possède, collectivement, toutes les compétences requises pour développer le produit.

Essaimage : organisation de l'équipe pendant le sprint qui consiste à se mettre à plusieurs pour finir vite une story.

Expert : personne, à l'extérieur de l'équipe, qui possède une compétence nécessaire à l'équipe pour achever une ou plusieurs stories d'un sprint.

Feature : service ou fonction d'un produit dont l'énoncé est clair pour les parties prenantes ; une feature contribue à un impact et se décompose en stories.

Impact : changement dans le comportement d'une partie prenante.

Impact mapping : technique de planification stratégique, basée sur une carte heuristique, qui aligne les travaux de l'équipe de développement avec les impacts attendus.

Incrément de produit : résultat d'un sprint.

Limite : pratique Kanban consistant à limiter le travail en cours. L'atteinte de la limite haute bloque une nouvelle entrée, celle de la limite basse déclenche un réapprovisionnement.

Kanban : méthode d'amélioration de processus apportant à Scrum la notion de flux.

Lean Startup : démarche de développement produit basé sur une approche pragmatique et itérative d'apprentissage par les retours des utilisateurs.

Mêlée quotidienne : moment de partage quotidien entre les membres de l'équipe permettant d'organiser la journée et de vérifier l'alignement sur l'objectif du sprint.

Objectif de la release : bénéfice attendu à la fin de la release.

Objectif du sprint : bénéfice attendu à la fin du sprint sur lequel s'engage l'équipe et qui détermine le succès du sprint.

Obstacle : fait concret qui bloque ou ralentit le bon déroulement d'une ou plusieurs tâches du sprint empêchant l'équipe d'avancer à son rythme.

Ordre (ou priorité ordinaire) : rang définissant la séquence de réalisation des stories

Partie prenante : toute personne intéressée par les résultats de l'équipe.

Persona : représentation fictive de l'utilisateur final d'un produit ou d'un service.

Plan de release : présentation des sprints à venir et du contenu prévu de ces sprints.

Plan de sprint : présentation des stories et leurs tâches sélectionnées pour un sprint.

Planning poker : séance d'estimation en groupe, avec des cartes, qui combine le jugement d'expert et l'estimation par analogie.

Point de story : estimation de la taille relative d'une story.

Pratique : approche concrète et éprouvée qui permet de résoudre un ou plusieurs problèmes courants ou d'améliorer la façon de travailler.

Produit : résultat du travail de l'équipe en vue d'impacter les parties prenantes.

Product Owner : personne de l'équipe ayant délégation des parties prenantes pour maximiser la valeur, et imputable du résultat auprès d'elles.

Référent de la story : lors de l'essaimage, développeur qui diffuse la connaissance sur la story et s'assure de sa finition.

Release : période de temps constituée d'une série de sprints aboutissant généralement à déployer une version du produit.

Revue : réunion pour collecter le feedback des parties prenantes et prendre une décision sur la vie du produit.

Rétrospective : réunion pendant laquelle l'équipe s'appuie sur le passé pour préparer le futur.

Roadmap : présentation à long terme, montrant plusieurs releases successives avec leurs objectifs.

Scrum : cadre de processus qui aide les gens à améliorer leur façon de travailler.

ScrumMaster : personne au service de l'équipe, pour l'aider à réaliser les travaux demandés par le Product Owner, en appliquant Scrum dans le contexte de l'organisation.

Scrum de scrums : pratique de Scrum à plusieurs équipes qui collaborent au même produit.

Spike : story qui est une étude permettant d'obtenir une réponse à un problème technique à propos d'une user story.

Sprint : bloc de temps fixé aboutissant à créer un incrément du produit potentiellement livrable.

Sprint zéro : période de temps après le début de la release, permettant de préparer le premier sprint.

Story : élément du backlog, apportant de la valeur à une partie prenante ou à l'équipe.

Story de finition de feature : story incluant le travail restant à faire pour finir une feature dont les stories sont finies.

Story de finition de sprint : story revenant à chaque sprint, incluant le travail à faire pour obtenir un incrément de sprint fini.

Story de réserve pour urgences : story ajoutée à chaque sprint pour traiter les urgences, dont le contenu n'est pas défini et dont la taille est basée sur les statistiques des sprints

précédents.

Story mapping : technique spatiale d'ordonnancement des features et de décomposition en stories.

Story technique : story qui apporte de la valeur à l'équipe.

Storyotype : ensemble de stories partageant la même liste de vérifications de fini et de prêt.

Tâche : travail contribuant à une story, n'apporte pas de valeur par lui-même.

Taille (de story ou feature) : attribut estimé (en points) indiquant quelle part de backlog sera réalisée.

User story : story qui apporte de la valeur à une ou plusieurs parties prenantes qui utilisent le produit.

Tableau de sprint : sert à montrer l'avancement des travaux pendant le sprint, c'est une représentation physique du plan de sprint.

Tableau de features : représentation des features dans leur état courant.

Test d'acceptation : test permettant d'accepter une user story du point de vue de son comportement.

Valeur métier : attribut estimé mesurant ce que combien quelque chose (story, feature, produit) apporte aux parties prenantes.

Vélocité : utilisée pour prévoir la capacité, mesure de la partie du backlog réalisée par l'équipe pendant un sprint.

Vision : expression d'une volonté collective de développer un excellent produit ou service.

Index

3A, [274](#)
3C, [143](#)
6D, [164](#)
acteur, [237](#)
affinage, [135](#), [159](#)
 du backlog, [296](#)
Agile Fluency, [231](#), [295](#)
agilité, [20](#), [297](#)
Andressen Marc, [21](#)
approche agile, [48](#)
approche itérative et incrémentale, [37](#)
bac à glace, [166](#), [172](#), [298](#)
bac à sable, [149](#), [165](#), [299](#)
bac d'affinage, [150](#), [300](#)
bac d'arrivée, [152](#), [302](#)
bac de départ, [151](#), [301](#)
backlog, [7](#), [100](#), [118](#), [121](#), [125](#), [303](#)
 affinage, [160](#)
 élément, [137](#)
 priorité, [241](#)
 purger, [171](#)
BDD, [256](#)
Beauregard François, [94](#)
bloc de temps, [40](#)
bug, [139](#), [156](#), [187](#), [286](#), [287](#), [304](#)
bugtracker, [157](#)
build, [282](#)
burndown chart, [273](#), [305](#)
 de sprint, [207](#)

burnup, [276](#)
burnup chart, [306](#)
cadre de processus, [4](#)
cadre Scrum, [221](#)
capacité, [260](#), [307](#)
capitalisation, [294](#)
carnet, [28](#), [123](#)
carte, [144](#)
changement, [127](#)
chef de projet, [106](#)
Cokburn Alistair, [27](#)
condition d'acceptation, [308](#)
confirmation, [146](#)
conflit entre personnes, [114](#)
contexte du projet, [228](#)
conversation, [145](#)
courage, [65](#)
culture backlog, [128](#)
cycle de développement, [31](#), [46](#)
Cynefin, [24](#)
définition de fini, [132](#), [310](#)
définition de prêt, [131](#), [163](#), [309](#)
Definition of Done, [179](#)
délai d'immersion, [195](#)
demi de mêlée, [109](#)
démonstration, [210](#)
dépendance, [158](#)
dettre technique, [141](#), [184](#), [186](#), [284](#), [311](#)
développement itératif, [14](#)
développeur, [75](#), [312](#)
DevOps, [54](#)

directeur de produit, [96](#)
document de spécification, [120](#)
Dupagne Dominique, [107](#)
engagement, [66](#)
epic, [133](#), [313](#)
équipe, [110](#), [191](#), [200](#), [205](#), [218](#)

- auto-organisation, [117](#)
- auto-organisée, [315](#)
- organisation, [77](#)
- pluridisciplinaire, [79](#), [317](#)

Scrum, [316](#)
taille, [76](#)
essaimage, [84](#), [194](#), [196](#), [318](#)
estimation, [254](#)

- de la taille, [278](#)
- de la valeur, [279](#)

expert, [91](#), [319](#)
Extreme Programming, [16](#)
feature, [136](#), [240](#), [320](#)
feedback, [53](#), [126](#), [214](#)
focus, focalisation, [62](#)
grooming, [161](#)
Highsmith Jim, [22](#)
histogramme de livraison restante, [29](#)
iceScrum, [93](#)
impact, [321](#)
impact mapping, [226](#), [232](#), [235](#), [239](#), [252](#), [269](#), [281](#), [322](#)
impediments, [113](#)
incrément de produit, [52](#), [178](#), [211](#), [323](#)
indicateurs, [271](#)
itération, [38](#)

jalon, [33](#), [43](#)

Jeffries Ron, [142](#)

Kanban, [17](#), [225](#), [292](#), [325](#)

Kniberg Henrik, [177](#)

KPI, [272](#)

Lean Software, [18](#)

Lean Startup, [57](#), [249](#), [326](#)

learning value, [90](#)

limite, [324](#)

loi

de Fitts, [245](#)

de Miller, [247](#)

management visuel, [11](#)

Manifeste agile, [12](#), [227](#)

Marick Brian, [95](#)

médiateur, [115](#)

mêlée quotidienne, [327](#)

méthode agile, [15](#), [39](#)

métier, [101](#)

MMF (Minimal Marketable Feature), [154](#)

modèle

Cynefin, [23](#)

de cycle, [34](#)

de Tuckman, [80](#)

mou (dans les plans), [201](#)

MVP (Minimum Viable Product), [58](#)

Nonaka Ikujiro, [10](#)

objectif

de la release, [328](#)

du sprint, [329](#)

obstacle, [111](#), [112](#), [203](#), [206](#), [330](#)

OOPSLA, [1](#)

open source, [270](#)

ordre, [331](#)

ouverture, [63](#)

pair programming, [85](#)

parties prenantes, [30](#), [59](#), [69](#), [236](#), [333](#)

profil, [88](#)

persona, [238](#), [334](#)

phase, [32](#), [47](#)

plan

de release, [335](#)

de sprint, [336](#)

planning poker, [25](#), [168](#), [175](#), [261](#), [337](#)

plateau projet, [82](#)

PO, [72](#), [97](#)

point de story, [338](#)

PP, [68](#)

pratiques, [13](#), [222](#), [339](#)

principe du [Z](#), plus ou moins [2](#), [246](#)

priorité, [124](#)

ordinale, [332](#)

process framework, [5](#)

Process Owner, [86](#)

processus, [35](#)

Product Owner, [71](#), [92](#), [169](#), [341](#)

disponibilité, [102](#)

équipe, [103](#)

responsabilités, [98](#)

produit, [340](#)

projet, [291](#)

purger, [170](#)

Rawsthorne Dan, [67](#)

refactoring, [283](#)

référent de la story, [198](#), [342](#)

référentiel des exigences, [122](#)

refinement, [162](#)

release, [45](#), [183](#), [343](#)

 à périmètre fixé, [268](#)

 fin, [50](#), [188](#)

 objectif, [263](#)

respect, [64](#)

rétrospective, [215](#), [217](#), [345](#)

réunion

 d'affinage, [176](#)

 de planification du sprint, [190](#)

revue, [89](#), [344](#)

 de backlog, [223](#), [251](#)

 de sprint, [209](#), [212](#)

roadmap, [346](#)

RTF (Running Tested Feature), [155](#)

salle dédiée, [83](#)

Schwaber Ken, [2](#)

Scrum, [8](#), [229](#), [347](#)

 cadre, [220](#)

 de scrums, [349](#)

 équipe, [70](#)

Scrum User Group, [116](#)

ScrumMaster, [73](#), [105](#), [348](#)

 responsabilités, [61](#), [108](#), [233](#)

Shu Ha Ri, [26](#), [230](#), [293](#)

SM, [74](#)

spécification, [119](#)

spike, [253](#), [285](#), [350](#)

sprint, [6](#), [36](#), [181](#), [257](#), [351](#)

burnup, [275](#)

de stabilisation, [51](#)

durée, [42](#)

exécution, [202](#)

fin, [44](#)

rétrospective, [216](#)

revue, [208](#)

zéro, [49](#), [173](#), [352](#)

stakeholders, [60](#)

stand up meeting, [204](#)

stories, [56](#)

story, [130](#), [353](#)

de découverte, [174](#)

de finition de feature, [354](#)

de finition de sprint, [355](#)

de finition du sprint, [182](#)

de réserve, [167](#), [288](#), [356](#)

épique, [134](#), [314](#)

fonctionnelle, [138](#)

référent, [197](#)

technique, [140](#), [358](#)

workflow, [148](#)

story map, [250](#)

story mapping, [357](#)

storyotype, [180](#), [189](#), [255](#), [359](#)

Sutherland Jeff, [3](#)

tableau, [264](#)

de features, [365](#)

de sprint, [193](#), [364](#)

tâche, [192](#), [199](#), [360](#)

taille

de feature, [362](#)

de story, [361](#)

Takeuchi Hirotaka, [9](#)

technical debt, [185](#)

Technical Owner, [87](#)

télétravail, [81](#)

temps

de cycle, [290](#)

de traversée, [289](#)

test, [55](#)

d'acceptation, [104](#), [366](#)

théorie de la Gestalt, [244](#)

timebox, [41](#), [258](#)

timeline, [219](#)

user story, [129](#), [363](#)

utilisabilité, [243](#)

utilité, [242](#)

ordinale, [280](#)

UX design, [248](#)

valeur métier, [367](#)

vélocité, [213](#), [259](#), [277](#), [368](#)

basse, [266](#)

maximum, [265](#)

Villepreux Pierre, [78](#)

vision, [99](#), [234](#), [369](#)

wall estimation, [262](#)

workflow

feature, [153](#)

story, [147](#)

XP, [19](#), [224](#)

zone d'incertitude, [267](#)

Table des matières

[Preface](#)

[Avant-propos](#)

[Chapitre 1. Scrum dans le mouvement agile](#)

[1.1. Premiers pas avec Scrum](#)

[1.2. Le mouvement agile](#)

[1.3. Scrum aujourd’hui](#)

[Chapitre 2. Le cycle des sprints](#)

[2.1. Approche itérative et incrémentale](#)

[2.2. Cycle de développement Scrum](#)

[2.3. Les sprints et releases sur le terrain](#)

[Chapitre 3. Les gens de Scrum](#)

[3.1. Importance des gens](#)

[3.2. L'équipe Scrum](#)

[3.3. Le développeur](#)

[3.4. Les parties prenantes](#)

[3.5. Les experts](#)

[Chapitre 4. Le rôle du Product Owner](#)

[4.1. Responsabilités du Product Owner](#)

[4.2. Compétences souhaitées](#)

[4.3. Choisir le Product Owner d'une équipe](#)

[4.4. Une journée typique de PO](#)

[4.5. Le PO sur le terrain](#)

[Chapitre 5. Le rôle du ScrumMaster](#)

[5.1. Responsabilités du ScrumMaster](#)

[5.2. Compétences souhaitées](#)

[5.3. Choisir le ScrumMaster d'une équipe](#)

[5.4. Une journée typique de SM](#)

[5.5. Le SM sur le terrain](#)

[Chapitre 6. Structurer le backlog](#)

[6.1. Un outil essentiel pour l'équipe](#)

[6.2. Hiérarchie des éléments de backlog](#)

[6.3. Types de stories](#)

[6.4. Parties du backlog](#)

[6.5. Tableau de features](#)

[6.6. Le backlog sur le terrain](#)

[Chapitre 7. Affiner le backlog](#)

[7.1. Définition de prêt](#)

[7.2. L'affinage, une pratique d'équipe](#)

[7.3. Les activités d'affinage](#)

[7.4. Affinage pendant le sprint zéro](#)

[7.5. Résultat de l'affinage](#)

[7.6. L'affinage sur le terrain](#)

Chapitre 8. La définition de fini

- 8.1. Finir l'incrément de produit
- 8.2. Finir les stories et les features
- 8.3. Finir le sprint et la release
- 8.4. Les activités pour définir fini et prêt
- 8.5. La définition de fini sur le terrain

Chapitre 9. Planifier le sprint

- 9.1. Les activités de planification du sprint
- 9.2. Activités de planification du sprint
- 9.3. Résultats de la planification du sprint
- 9.4. La « planif » sur le terrain

Chapitre 10. La mêlée quotidienne

- 10.1. Suivre l'exécution du sprint
- 10.2. Une réunion quotidienne
- 10.3. La mêlée classique
- 10.4. La mêlée orientée stories
- 10.5. Les informations utiles au quotidien
- 10.6. La mêlée sur le terrain

Chapitre 11. La revue de sprint

- 11.1. Plus qu'une démo
- 11.2. Les activités de la revue de sprint
- 11.3. La revue sur le terrain

Chapitre 12. La rétrospective

- 12.1. Une pratique d'amélioration continue
- 12.2. Les activités de la rétrospective
- 12.3. Les résultats de la rétrospective
- 12.4. La rétrospective sur le terrain

Chapitre 13. Contextualiser Scrum

- 13.1. Pratiques agiles
- 13.2. Caractériser le contexte
- 13.3. Étudier l'impact sur les pratiques
- 13.4. Adapter en fonction de la situation
- 13.5. La contextualisation sur le terrain

Chapitre 14. Découvrir le produit

- 14.1. De l'idée aux features
- 14.2. Définir la vision produit/vision
- 14.3. Identifier les parties prenantes
- 14.4. Définir le produit attendu en fin de release
- 14.5. La découverte du produit sur le terrain

Chapitre 15. Raconter la story

- 15.1. Identifier des stories avec le story mapping
- 15.2. Décomposer
- 15.3. Ajouter une condition d'acceptation
- 15.4. Accepter la story

Chapitre 16. Planifier la release

- 16.1. Pourquoi planifier plus loin que le sprint ?

- [16.2. Les bases de la planification de release](#)
- [16.3. Les activités de planification de release](#)
- [16.4. Engagement sur le plan de release](#)
- [16.5. Résultats de la planification de release](#)
- [16.6. La planification de release sur le terrain](#)

[Chapitre 17. Tirer profit des outils](#)

- [17.1. Les Post-it®](#)
- [17.2. Les outils informatiques](#)
- [17.3. Les tableaux](#)
- [17.4. Les outils et les tableaux sur le terrain](#)
- [17.5. Les jeux](#)
- [17.6. Les jeux sur le terrain](#)

[Chapitre 18. Améliorer la visibilité avec des indicateurs](#)

- [18.1. Indicateurs pour le suivi du sprint](#)
- [18.2. Indicateurs relatifs à l'équipe](#)
- [18.3. Indicateurs pour le suivi de la release](#)
- [18.4. Pas d'indicateur de productivité](#)
- [18.5. Pas d'indicateur du niveau d'agilité](#)
- [18.6. Les indicateurs sur le terrain](#)

[Chapitre 19. Ajouter les pratiques de développement XP](#)

- [19.1. Pratiques autour du code](#)
- [19.2. Pratiques de conception](#)
- [19.3. Maintenance](#)
- [19.4. Pratiques de développement sur le terrain](#)

[Chapitre 20. Appliquer Kanban sur Scrum](#)

- [20.1. Pourquoi Kanban sur Scrum ?](#)
- [20.2. Limiter les tâches](#)
- [20.3. Limiter les stories](#)
- [20.4. Limiter les features](#)
- [20.5. Mesures et indicateurs](#)
- [20.6. Arrêter Scrum pour Kanban ?](#)

[Chapitre 21. Développer un produit avec plusieurs équipes](#)

- [21.1. Un projet Scrum ?](#)
- [21.2. Cycle de vie produit](#)
- [21.3. Les gens avec plusieurs équipes](#)
- [21.4. Backlog et affinage à plusieurs équipes](#)
- [21.5. Les événements du sprint à l'échelle](#)
- [21.6. Planification à grande échelle](#)
- [21.7. Scrum à plusieurs équipes sur le terrain](#)

[Chapitre 22. Transformer les organisations](#)

- [22.1. Pourquoi se transformer ?](#)
- [22.2. Comment se transformer ?](#)
- [22.3. Trois obstacles de nos organisations](#)
- [22.4. Des outils pour la transformation](#)

[Quiz](#)

[Glossaire](#)

[Index](#)