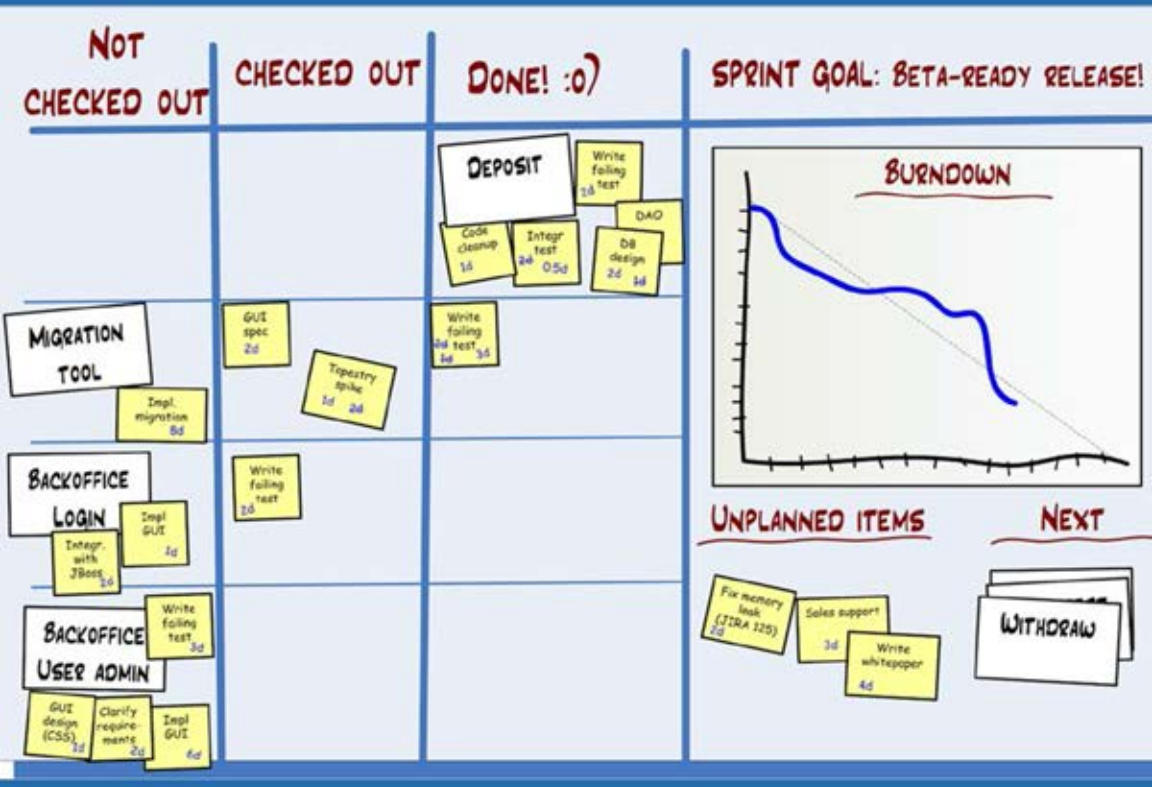
2nde Edition – Version du Réalisateur



SCRUM ET XP DEPUIS LES TRANCHEES

Comment nous appliquons Scrum

Henrik Kniberg

InfoQueue
ENTERPRISE SOFTWARE DEVELOPMENT SERIES

Scrum and XP from the Trenches – 2nd Edition

Published by C4Media, publisher of InfoQ.com.

Production Editor: Ana Ciobotaru
Copyeditor: Professor Laurie Nyveen
Cover and Interior Design: Dragos Balasoiu

Library of Congress Cataloguing-in-Publication Data:

ISBN: 978-1-4303-2264-1

Printed in the United States of America

Remerciements

Le premier brouillon de ce papier a été tapé en un week-end seulement, mais ce fut un sacré week-end ! Focalisation à 150% :o)

Merci à ma femme Sophia et mes enfants Dave et Jenny pour avoir supporté mon asocialité ce week-end, et aux parents de Sophia, Eva et Jörgen, pour être venus aider à prendre en charge la famille.

Merci également à mes collègues de Crisp à Stockholm et aux gens sur le groupe yahoo scrumdevelopment pour avoir corrigé le papier et pour m'avoir aidé à l'améliorer.

Et, finalement, merci à tous mes lecteurs qui ont fourni un flux constant de feedback utile. Je suis particulièrement content d'entendre que ce papier a incité autant d'entre vous à essayer le développement de logiciels agile !

Table des matières

Préface de Jeff Sutherland

Les équipes doivent connaître les bases de Scrum. Comment est-ce que vous créez et estimez un backlog de produit ? Comment le transformez-vous en backlog d'itération ? Comment gérez-vous une courbe du reste à faire et calculez la vélocité de l'équipe ? Le livre d'Henrik est un kit de démarrage avec les pratiques de base qui aident les équipes à passer d'un essai de Scrum à une bonne implémentation de Scrum.

Une bonne implémentation de Scrum devient plus importante pour les équipes qui cherchent des investissements financiers. En tant que coach Agile pour un groupe investissant en capital-risque, je les aide à investir seulement dans les entreprises agiles qui appliquent bien les pratiques agiles.

L'associé principal du groupe a demandé à toutes les sociétés au sein de son portefeuille si elles connaissent la vélocité de leurs équipes. Elles ont des difficultés à répondre à la question tout de suite. Les futures opportunités d'investissement vont obliger les équipes à comprendre leur vélocité de développement logiciel.

Pourquoi est-ce si important? Si les équipes ne connaissent pas leur vélocité, le Directeur de produit ne peut créer de feuille de route pour le produit avec des échéances crédibles. Sans des échéances fiables, l'entreprise peut échouer et les investisseurs peuvent perdre leur argent !

Ce problème touche les grandes sociétés comme les petites, nouvelles ou anciennes, financée ou non. Lors d'une discussion récente sur l'implémentation de Scrum de Google à une conférence londonienne, j'ai demandé à une audience de 135 personnes combien appliquaient Scrum et 30 ont répondu positivement. Je leur ai ensuite demandé s'ils faisaient du développement itératif selon les standards de Nokia. Le développement itératif est fondamental dans le Manifeste

Agile - délivrer tôt et fréquemment du logiciel qui marche. Après des années de rétrospectives avec des centaines d'équipes Scrum, Nokia a développé des exigences de base pour le développement itératif :

- Les itérations doivent être de durée fixe et inférieure à six semaines.

- Le code à la fin de l'itération doit être testé par le AQ et doit fonctionner correctement.

Sur les 30 personnes qui prétendaient appliquer Scrum, seulement la moitié affirmait se conformer au premier principe du Manifeste Agile sur les standards Nokia. J'ai ensuite demandé s'ils suivaient les standards Nokia pour Scrum :

- Une équipe Scrum doit avoir un Directeur de produit et doit savoir qui c'est.
- Le Directeur de produit doit gérer un Backlog de produit avec des estimations fournies par l'équipe.
- L'équipe doit avoir une Courbe du reste à faire et doit connaître sa vélocité.
- Aucune personne extérieure ne doit interférer avec l'équipe durant le Sprint.

L'apport du livre d'Henrik est que, si vous suivez les pratiques décrites, vous aurez un Directeur de produit, des estimations pour votre Backlog de produit, une Courbe du reste à faire, et vous connaîtrez la vélocité de votre équipe ainsi que de nombreuses autres pratiques essentielles pour un Scrum dangereusement opérationnel. Vous passerez le test Nokia pour Scrum et serez digne de l'investissement dans votre travail. Si vous êtes une startup, vous pouvez même bénéficier du financement d'une société capital-risque. Vous serez peut-être le futur du développement logiciel et le créateur d'une nouvelle génération d'éminents logiciels.

Jeff Sutherland,
Ph.D., Co-Créateur de Scrum

Préface de Mike Cohn

Scrum et Extreme Programming (XP) demandent tous deux aux équipes de finir certains éléments tangibles et livrables de leur travail à la fin de chaque itération. Ces itérations sont conçues pour être courtes et à durée finie. L'accent mis sur le fait de livrer du code qui marche au bout d'une courte période de temps signifie que les équipes Scrum et XP n'ont pas de temps pour faire de la théorie. Elles ne s'évertuent pas à dessiner le modèle UML parfait dans leur outil de modélisation, à écrire des cahiers des charges parfaits, ou à écrire du code qui pourra s'adapter à tous les changements futurs imaginables. A la place, les équipes Scrum et XP se concentrent pour que les choses soient terminées. Ces équipes acceptent qu'elles puissent faire des erreurs en chemin, mais elles réalisent aussi que le meilleur moyen pour trouver ces erreurs est d'arrêter de penser au logiciel au niveau théorique de l'analyse et de la conception, mais plutôt de se lancer, de se salir les mains, et de commencer à construire le produit.

C'est par cette focalisation sur le « faire » plutôt que le « théoriser » que se distingue ce livre. Il est facile de s'apercevoir que Henrik Kniberg l'a bien compris dès le début. Il ne fait pas de longs discours sur ce qu'est Scrum ; pour ça, il se réfère à des sites faciles à comprendre. Au lieu de cela, Henrik entre dans le vif du sujet et commence directement à décrire comment son équipe gère et travaille son carnet de produit. A partir de là, il parcourt tous les autres éléments et les pratiques d'un projet agile bien-portant. Pas de théorie. Pas de référence. Pas de note de bas de page. Aucun besoin de cela. Le livre d'Henrik n'est pas un exposé philosophique expliquant pourquoi Scrum marche ou

pourquoi vous voudriez utiliser ceci ou cela. Il s'agit d'une description sur comment une certaine équipe agile efficace travaille.

C'est pour cette raison que le sous-titre du livre, « Comment nous appliquons Scrum », est si approprié. Ce n'est peut-être pas la façon dont vous appliquez Scrum, c'est comment l'équipe d'Henrik applique Scrum. Il se peut que vous vous demandiez pourquoi vous devriez vous intéresser à la façon dont une autre équipe applique Scrum. Vous devriez vous y intéresser car nous pouvons tous apprendre à améliorer notre pratique de Scrum en tirant partie des expériences des autres, spécialement lorsqu'ils le font bien. Il n'y a pas et il n'y aura jamais de liste des « meilleures pratiques Scrum » car le contexte de l'équipe et du projet prévaut sur toute autre considération. A la place, ce que nous avons besoin de savoir, ce sont les bonnes pratiques et les contextes dans lesquels elles ont réussi. Lisez assez de témoignages d'équipes qui réussissent et comment elles se sont débrouillées et vous serez préparé pour affronter les obstacles survenus sur votre chemin durant votre mise en œuvre de Scrum et XP.

Henrik fournit une multitude de bonnes pratiques ainsi que l'indispensable contexte pour nous aider à mieux comprendre comment appliquer Scrum et XP dans les tranchées de nos propres projets.

Mike Cohn
Auteur de *Agile Estimating and Planning* et *User Stories Applied for Agile Software Development.*

Préface - Hé, Scrum ça marche !

Scrum ça marche ! Au moins pour nous (c'est-à-dire mon client actuel à Stockholm, dont je n'ai pas l'intention de mentionner le nom ici). J'espère qu'il marchera pour vous aussi ! Peut-être que ce papier vous aidera au long du chemin.

C'est la première fois que j'ai vu une méthodologie de développement (désolé Ken, un *cadre*) marcher exactement comme dans le bouquin. Plug 'n play. Nous sommes tous heureux avec lui – développeurs, testeurs, managers. Il nous a aidés à nous sortir d'une situation difficile, et nous a permis de maintenir une focalisation et un élan malgré de sévères turbulences du marché et des réductions de personnel.

Je ne devrais pas dire que j'ai été surpris, mais, eh bien oui, je l'ai été. Après avoir initialement digéré quelques livres sur le sujet Scrum semblait bien, mais près trop bien pour être vrai (et nous connaissons tous le proverbe « quand quelque chose semble trop beau pour être vrai… »). Donc j'étais légitimement un peu sceptique. Mais après avoir appliqué Scrum pendant un an je suis suffisamment impressionné (et la plupart des gens dans les équipes aussi) pour que je continue probablement à utiliser Scrum par défaut dans les nouveaux projets tant qu'il n'y a pas de bonnes raisons de faire autrement.

Préface – 2nde édition

Huit ans sont passés, et ce livre est toujours très populaire. Ouah ! Je n'aurais jamais pu imaginer l'impact que ce petit livre aurait ! Je tombe encore sur des équipes, managers, coaches, et formateurs de partout qui l'utilisent comme leur guide principal pour le développement logiciel agile.

Mais le fait est que, j'ai beaucoup appris depuis 2007 ! Alors le livre a vraiment besoin d'une mise à jour.

Depuis la publication du livre, j'ai eu l'opportunité de travailler avec de nombreux leaders visionnaires agiles et lean ; certains sont même devenus comme des mentors pour moi. Un merci tout particulier à Jeff Sutherland, Mary et Tom Poppendieck, Jerry Weinberg, Alistair Cockburn, Kent Beck, Ron Jeffries, et Jeff Patton – Je ne pourrais imaginer un meilleur groupe de conseillers !

J'ai également eu la chance d'aider beaucoup de sociétés à implémenter ces idées en pratique : des sociétés en crise mais aussi des sociétés très prospères voulant devenir encore meilleures. Au final, ce fut un voyage assez hallucinant !

Lorsque je relis ce vieux livre, je suis surpris par toutes les choses avec lesquelles je suis encore d'accord. Mais il y a également des pages que j'aimerais déchirer et dire « Bordel, à quoi je pensais ? Ne le fais pas comme ça ! Il y a une bien meilleure façon ! »

Vu que le livre est une étude de cas réels, je ne peux pas changer l'histoire. Ce qu'il s'est passé est ce qu'il s'est passé. Mais je peux émettre des commentaires !

Alors c'est ce que la 2nde édition est – une version annotée du livre original. Comme une version du réalisateur. Voyez cela comme moi debout derrière votre épaule pendant que vous lisez le livre, commentant des éléments, vous réconfortant, avec l'occasionnel rire et grognement.

Voici ce à quoi les annotations ressemblent. Tout le reste (à l'exception de cette préface) est le livre original, non modifié, et ces boites colorées sont mes commentaires et réflexions pour la seconde édition.

J'introduirais également quelques exemples d'autres sociétés, principalement Spotify (vu que c'est là où j'ai passé la plupart de mon temps dernièrement), mais aussi d'autres endroits. Amusez-vous bien !

Henrik, Mars 2015

1

Intro

Vous êtes sur le point d'utiliser Scrum dans votre organisation. Ou peut-être vous avez déjà utilisé Scrum pendant quelques mois. Vous avez les bases, vous avez lu les livres, peut-être que vous avez même obtenu votre certification de Scrum Master. Félicitations !

Mais pourtant vous ressentez de la confusion.

Selon les mots de Ken Schwaber, Scrum n'est pas une méthodologie, c'est un *cadre*. Ce qui signifie que Scrum ne va pas réellement vous dire exactement ce que vous devez faire. Zut.

La bonne nouvelle c'est que **je** vais vous dire comment **je** pratique Scrum, dans les plus petits détails. La mauvaise nouvelle est qu'il s'agit uniquement de comment **je** pratique Scrum. Cela ne signifie pas que *vous* devriez faire exactement la même chose. En fait **je** pourrais même le faire d'une autre manière si **je** rencontrais une autre situation.

La force et la douleur de Scrum est que vous êtes obligés de l'adapter à votre situation particulière.

Mon approche actuelle de Scrum est le résultat d'une année d'expérimentation dans une équipe de développement d'environ 40 personnes. L'entreprise était dans une situation difficile avec beaucoup d'heures de travail supplémentaires, de graves problèmes de qualité, des « incendies » à éteindre constamment, des dates de livraison manquées, etc. L'entreprise avait décidé d'utiliser Scrum, mais n'avait pas terminé son implémentation, ce qui devait être mon travail. Pour la plupart des membres de l'équipe de développement à l'époque, Scrum était juste un étrange mot à la mode entendu de temps en temps dans les couloirs, sans réelle implication sur leur travail quotidien.

Sur une année nous avons implémenté Scrum à travers toutes les couches de l'entreprise, essayé différentes tailles d'équipe (3-12 membres), différentes durées d'itération (2-6 semaines), différentes manières de définir « terminé », différents

formats pour les carnets de produit et les carnets de d'itération (Excel, Jira, cartes d'index), différentes stratégies de test, différentes façons de faire les démonstrations, différentes manières de synchroniser des équipes Scrum multiples, etc. Nous avons également expérimenté avec les pratiques XP - différentes manières de faire l'intégration continue, la programmation en binôme, le développement dirigé par les tests, etc., et comment les combiner avec Scrum.

C'est un processus d'apprentissage constant, donc l'histoire ne s'arrête pas ici. Je suis convaincu que cette entreprise va continuer d'apprendre (s'ils poursuivent les rétrospectives de fin d'itération) et de découvrir de nouvelles idées sur les meilleures manières d'implémenter Scrum dans leur contexte particulier.

Avertissement

Ce document ne prétend pas représenter la « bonne » manière de pratiquer Scrum. Il représente seulement une façon de pratiquer Scrum, le résultat d'une année de perfectionnement constant. Vous pourriez même décider que nous avons tout faux.

Tout le contenu de ce document reflète mes opinions personnelles et subjectives, et n'est en aucune façon une déclaration officielle de Crisp ou de mon client actuel. Pour cette raison j'ai intentionnellement évité de mentionner des personnes ou des produits spécifiques.

Pourquoi j’ai écrit ceci

En me formant à Scrum j'ai lu les livres pertinents sur Scrum et l'agilité, je me suis épanché dans les sites et forums sur Scrum, j'ai suivi la certification de Ken Schwaber, je l'ai mitraillé de questions, et j'ai passé beaucoup de temps à discuter avec mes collègues. Toutefois l'une de mes meilleures sources d'information a été les vraies histoires de guerre. Les histoires de guerre transforment les Principes et Pratiques en ... eh bien ... Comment faire en pratique. Elles m'ont aussi permis d'identifier (et parfois d'éviter) les erreurs typiques de débutant.

Alors c'est l'occasion pour moi de donner quelque chose en retour. Voici mon histoire de guerre.

J'espère que ce papier va susciter un feedback utile de la part de ceux qui sont dans la même situation. Eclairez-moi s'il vous plaît !

Mais Scrum, qu'est-ce que c'est ?

Oh, désolé. Vous êtes complètement débutant en Scrum et XP ? Dans ce cas vous pourriez avoir envie de jeter un œil aux liens suivants :

- http://agilemanifesto.org/

- http://www.mountaingoatsoftware.com/scrum
- http://www.xprogramming.com/xpmag/whatisxp.htm

Regardez le Guide Scrum également. C'est aujourd'hui la description officielle de Scrum, maintenue par Jeff Sutherland et Ken Schwaber.
http://www.scrumguides.org

Si vous êtes trop impatient pour le faire, sentez-vous libres de simplement continuer à lire. La plupart du jargon Scrum est expliqué au fur et à mesure, aussi vous pourriez quand même trouver cette lecture intéressante.

2

Comment nous faisons les backlogs de produit

Le Backlog de produit est le cœur de Scrum. C'est là où tout commence.

Euh, non, le Backlog de produit n'est pas le point de départ. Un bon produit démarre avec un besoin client et une vision de comment le satisfaire. Le Backlog de produit est le résultat du raffinement de cette vision en livrables concrets. Le voyage de la vision au Backlog peut être assez complexe, et beaucoup de techniques sont apparues pour combler ce manque. Des choses comme le user-story mapping (lisez le livre de Jeff Patton, il est génial !), lean UX, impact mapping, et plus. Mais n'utilisez néanmoins pas cela comme excuse pour faire une grosse conception initiale ! Laissez le Backlog de produit émerger itérativement, comme tout le reste.

Le Backlog de produit est en gros une liste priorisée d'exigences, d'histoires, de caractéristiques ou autre. Des choses que le client veut, décrites selon la terminologie du client.

Nous appelons ça des histoires, ou parfois simplement des *éléments du backlog*.

Nos histoires incluent les champs suivants :

- **ID** - un identifiant unique, juste un nombre auto-incrémenté. Cela évite de perdre la trace des histoires quand on les renomme.

- **Nom** - Un nom, une description courte de l'histoire. Par exemple « Voir l'historique de ses transactions ». Suffisamment claire pour que les développeurs et le directeur de produit comprennent approximativement de quoi ils parlent, et suffisamment claire pour la distinguer des autres histoires. Normalement 2 à 10 mots.

- **Importance** - l'importance attribuée par le directeur de produit à cette histoire. Par exemple 10. Ou 150. Élevée = plus important.

- Je fais attention d'éviter le terme « priorité » parce que la priorité 1 est typiquement considérée comme la plus « haute » priorité, ce qui est dommage si plus tard vous décidez que quelque chose d'autre est encore plus important. Quelle priorité devrait-on lui attribuer ? Priorité 0 ? Priorité -1 ?

- **Estimation initiale** - l'évaluation initiale de l'équipe sur la quantité de travail qui est nécessaire pour implémenter cette histoire par rapport aux autres histoires. L'unité est les points d'histoire et correspond habituellement à peu près à des « jours-hommes idéaux ».
  - Demandez à l'équipe « si vous pouvez prendre le nombre optimal de personnes pour cette histoire (ni trop peu ni trop nombreux, typiquement 2), et que vous vous enfermez dans une salle avec plein de nourriture et que vous travaillez sans jamais être dérangé, après combien de jours sortiriez-vous avec une implémentation finie, démontrable, testée, et livrable ? ». Si la réponse est « avec 3 gars enfermés dans une salle ça prendrait approximativement 4 jours » alors l'estimation initiale est de 12 points d'histoire.

  - Le point important n'est pas d'obtenir des estimations absolues correctes (c'est-à-dire que 2 points d'histoire devraient prendre 2 jours), le point important est d'avoir des estimations relatives correctes (c'est-à-dire que 2 points d'histoire devraient nécessiter à peu près la moitié du travail de 4 points d'histoire).

- **Comment le démontrer** - une description succincte de comment cette histoire sera démontrée à la démo de fin d'itération. C'est essentiellement la spécification d'un test simple. « Fais ci, ensuite fais ça, puis ceci devrait arriver ».
- Si vous pratiquez le DDT (Développement Dirigé par les Tests) cette description peut être utilisée comme du pseudo-code pour vos tests d'acceptance.
- **Notes** - n'importe quelle autre info, des clarifications, des références aux autres sources d'info, etc. Normalement très bref.

BACKLOG DE PRODUIT (exemple)					
ID	**Nom**	**Imp**	**Est**	**Démo.**	**Notes**
1	Dépôt	30	5	Authentification, ouvrir la page de dépôt, déposer 10€, aller sur la page du solde et vérifier que	Nécessite un diagramme de séquences UML. Ne pas se soucier du

				ça a bien augmenté de 10€.	cryptage pour l'instant.
2	Voir l'historique de ses transactions	10	8	Authentification, cliquez sur « transactions ». Faire un dépôt. Revenir aux transactions, vérifier que le nouveau dépôt apparaît.	Utiliser la pagination pour éviter des requêtes volumineuses . Conception semblable à la page des utilisateurs.

Nous avons expérimenté beaucoup d'autres champs, mais à la fin, les six champs ci-dessus étaient les seuls que nous utilisions sprint après sprint.

Il y a deux choses que je fais presque toujours différemment maintenant. Tout d'abord, il n'y a pas de colonne « importance ». A la place, j'ordonne juste la liste. Pratiquement tous les outils de gestion de Backlog ont des fonctionnalités de tri en glisser-déposer (même Excel et Google Spreadsheets, si vous apprenez la combinaison top secrète fondamentale). C'est plus facile et plus rapide. Ensuite, pas de jours-homme. Les estimations sont en points d'histoire ou en tailles de T-shirt (S/M/L), voire même pas du tout. Mais on verra ça plus tard.

Nous faisons habituellement un document Excel avec le partage activé (c'est-à-dire que plusieurs utilisateurs peuvent modifier le fichier en même temps). Officiellement le directeur de produit possède ce document, mais nous ne voulons pas verrouiller l'accès aux autres utilisateurs. Régulièrement un développeur veut ouvrir le document pour clarifier quelque chose ou changer une estimation.

Pour la même raison, nous ne plaçons pas le document dans le dépôt de contrôle de version ; nous le plaçons dans un répertoire partagé plutôt. Ceci semble la manière la plus simple d'autoriser les modifications simultanées sans causer des verrous ou des conflits de réconciliation.

Cependant, presque tous les autres artefacts sont placés dans le système de contrôle de version.

Excel, hein ? Ouah, c'était le vieux temps. Je n'envisagerais jamais aujourd'hui d'utiliser Excel pour la gestion de Backlog, à moins que ce soit une version cloud. Le Backlog de produit a besoin de vivre dans un document partagé en ligne qui soit accessible, facilement éditable, même simultanément comme n'importe lequel des milliers d'outils de gestion de Backlog disponibles (Trello, LeanKit et Jira sont populaires) ou un Google Spreadsheet (très pratique !).

Champs d'histoire supplémentaires

Parfois nous utilisons des champs supplémentaires dans le product backlog, surtout par commodité pour le directeur de produit pour l'aider à trier ses priorités.

- **Catégorie** - un classement approximatif de cette histoire, par exemple « back office » ou « optimisation ». De cette façon le directeur de produit peut facilement filtrer tous les éléments « optimisation » et leur affecter une priorité basse, etc.
- **Composants** - Habituellement représentés par des « cases à cocher » dans un tableau Excel, par exemple « base de données, serveur, client ». Ici l'équipe ou le directeur de produit peut identifier quels composants techniques vont être impliqués dans l'implémentation de cette histoire. Ceci est utile quand vous avez plusieurs équipes Scrum, par exemple une équipe back office et une équipe client, et que vous souhaitez rendre plus facile pour chaque équipe de décider quelles histoires prendre.
- **Demandeur** - le directeur de produit peut vouloir garder une trace de quel client ou quelle partie prenante avait demandé cet élément à l'origine, dans le but de les informer sur l'avancement.
- **ID de suivi de bug** - si vous avez un système de suivi des bogues séparé, comme nous faisons avec Jira, il est utile de garder une trace de toute correspondance directe entre une histoire et un ou plusieurs bogues.

Comment nous gardons le backlog de produit à un niveau métier

Si le directeur de produit a un passé technique il peut vouloir ajouter des histoires comme « ajouter des indexes sur les table d'événements ». Pourquoi veut-il faire ça ? Le but réel sous-jacent est probablement quelque chose du genre « augmenter la rapidité de la recherche des évènements sur le back office ».

Il se peut que les indexes n'étaient pas le goulot d'étranglement qui rendait le formulaire lent. Il se peut que ce soit quelque chose de complètement différent. L'équipe est normalement mieux placée pour deviner comment résoudre quelque chose, aussi le directeur de produit devrait garder le focus sur les objectifs métiers.

Quand je vois des histoires orientées technique comme celle-ci, je pose normalement au directeur de produit une série de questions « mais pourquoi » jusqu'à ce qu'on trouve le but sous-jacent.
Ensuite on reformule l'histoire dans les termes de ce but sous-jacent (« augmenter la rapidité de la recherche des évènements sur le back office »). La description technique d'origine finit en tant que note (« indexer la table des évènements peut peut-être résoudre le problème »).

Il y a un vieux modèle bien établi pour cela : « En tant que X, je veux Y, afin de Z. » Par exemple « En tant qu'acheteur, je veux sauvegarder mon caddie, afin de pouvoir continuer mes achats demain. » Je suis vraiment surpris de ne pas avoir eu vent de cela en 2007 ! Cela aurait été très commode. Oui, il y a des modèles plus élaborés disponibles aujourd'hui, mais l'utilisation du premier est un bon début, surtout pour les équipes qui sont nouvelles à l'Agile. Le modèle vous force à poser les bons types de questions, et réduit le risque d'être coincé par des détails techniques.

3

Comment nous préparons la réunion de planning du sprint

OK, le jour de planification du sprint arrive rapidement. Une leçon que nous réapprenons sans cesse est : assurez-vous que le backlog de produit soit en ordre *avant* cette réunion de planification du sprint.

Amen à cela ! J'ai vu beaucoup de réunions de planning de sprint exploser à cause d'un backlog de produit en bazar. Vous connaissez l'adage « merde en entrée = merde en sortie » ? Exactement.

Et qu'est-ce que *cela* signifie ? Que toutes les histoires doivent être parfaitement définies ? Que toutes les estimations doivent être correctes ? Que toutes les priorités doivent être fixées ? Non, non et non ! Tout ce que cela veut dire est :

- le backlog de produit devrait exister ! (imaginez que non ?)
- il devrait y avoir *un* backlog de produit et *un* directeur de produit (par produit bien sûr)
- un niveau d'importance devrait avoir été attribué à tous les éléments importants, et ces niveaux devraient être *différents*.
    - En fait, c'est OK si les éléments de plus faible importance ont le même niveau, puisqu'ils ne seront probablement pas mentionnés durant la planification de sprint de toute manière.
    - Si le directeur de produit croit qu'une histoire a une probabilité (même très faible) d'être adoptée dans la prochaine itération, alors cette histoire devrait avoir un niveau d'importance unique.
    - Le niveau d'importance est utilisé uniquement pour trier les éléments par importance. Donc si l'élément A a une importance de 20 et l'élément B une importance de 100, cela signifie simplement que B est plus important que A. Cela *ne signifie pas* que B est cinq fois plus important que A. Si B avait pour niveau d'importance 21, cela signifierait exactement la même chose !
    - Il est utile de laisser des trous dans la séquence des niveaux pour le cas où un élément C arrive et est plus important que A mais moins important que B. Bien sûr vous pourriez utiliser un

niveau d'importance de 20,5 mais cela devient déplaisant, donc nous laissons des trous à la place !

Mouais, triez juste la liste et vous n'aurez pas besoin de vous embêter avec les niveaux d'importance.

- le directeur de produit devrait *comprendre* chaque histoire (normalement il en est l'auteur, mais il arrive que d'autres personnes ajoutent des demandes, dont le directeur de produit peut choisir la priorité). Il n'a pas besoin de savoir exactement ce qui doit être implémenté, mais il devrait comprendre pourquoi l'histoire est là.

Note : d'autres personnes que le directeur de produit peuvent ajouter des histoires au backlog de produit. Mais elles n'ont pas à attribuer un niveau d'importance, seul le directeur de produit en a le droit. Elles n'ont pas à ajouter d'estimations non plus, seule l'équipe en a le droit.

Voici d'autres approches que nous avons essayées :

- Utiliser Jira (notre système de suivi de bugs) pour héberger le backlog de produit. Toutefois la plupart de nos directeurs de produit ont trouvé qu'il fallait trop de clics pour l'utiliser. Excel est agréable et facile à manipuler directement. Vous pouvez facilement utiliser des codes de couleur, réarranger les éléments, ajouter de nouvelles colonnes ad-hoc, ajouter des notes, importer et exporter des informations, etc.

Pareil avec Google Spreadsheets. Et c'est dans le cloud. Multi-utilisateurs, édition simultanée. Juste pour dire.

- Utiliser un outil de support de processus agile comme VersionOne, ScrumWorks, XPlanner, etc. Nous n'avons pas encore trouvé le temps d'en tester un, mais nous le ferons probablement.

4

Comment nous faisons les plannings de sprint

Le planning de sprint est une réunion critique, probablement l'événement le plus important dans Scrum (d'après mon opinion subjective bien sûr). Une réunion de planning de sprint mal exécutée peut perturber un sprint complet.

Important ? Oui. Evénement le plus important dans Scrum ? Non ! Les rétrospectives sont bieeen plus importantes ! Parce-que des rétrospectives qui fonctionnent bien vont aider à résoudre d'autres éléments endommagés. Le planning de sprint tend à être assez trivial tant que les autres éléments sont en place (un bon backlog de produit, un product owner et une équipe engagés, etc.) Aussi, sprinter n'est pas la seule façon d'être agile – beaucoup d'équipes utilisent kanban à la place. J'ai même écrit un mini-book à ce sujet : « Kanban et Scrum - tirer le meilleur des deux ».
http://www.infoq.com/fr/minibooks/kanban-scrum-minibook

Le but de la réunion de planning de sprint est de donner à l'équipe suffisamment d'informations pour qu'elle soit capable de travailler paisiblement, sans dérangements pendant quelques semaines, et de donner au directeur de produit suffisamment de confiance pour les laisser faire.

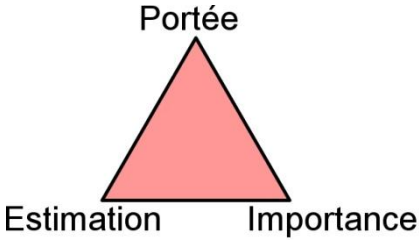
OK, c'est un peu flou. Concrètement, à la fin de cette réunion on doit avoir :

- Un but pour le sprint.
- Une liste des membres d'équipe (et de leur niveau d'engagement, si ce n'est pas 100%).
- Un backlog de sprint (= une liste des histoires incluses dans le sprint).
- Une date bien définie pour la démonstration.
- Une heure et un lieu bien définis pour la mêlée quotidienne.

Pourquoi le directeur de produit doit y assister

Quelquefois les directeurs de produits rechignent à passer des heures avec l'équipe pour faire le planning de sprint. « Les gars, j'ai déjà listé ce que je voulais. Je n'ai pas le temps de participer à votre réunion de planning ». C'est un problème vraiment sérieux.

La raison pour laquelle l'équipe au complet ainsi que le directeur de produit doivent être à cette réunion de planning de sprint est que chaque histoire contient trois variables qui dépendent fortement les unes des autres.



La portée et l'importance sont déterminées par le directeur de produit. L'estimation est déterminée par l'équipe. Durant une réunion de planning de sprint, ces trois variables sont affinées continuellement grâce au dialogue face-à-face entre l'équipe et le directeur de produit.

Normalement le directeur de produit commence la réunion en résumant son but pour le sprint et les histoires les plus importantes. Ensuite l'équipe parcourt chaque histoire et estime le temps qu'il faudra pour la réaliser, en commençant par la plus importante. Pendant qu'ils font ça, ils auront des questions importantes sur la portée – « est-ce que cette histoire de suppression d'utilisateur comprend le fait de parcourir chaque transaction en attente pour cet utilisateur, et de l'annuler? » Dans certains cas les réponses surprendront l'équipe et la conduiront à changer son estimation.

Dans d'autres cas l'estimation du temps pour une histoire ne sera pas ce que le directeur de produit attendait. Cela peut le conduire à changer l'importance de l'histoire. Ou à changer la portée de l'histoire, ce qui conduira l'équipe à ré-estimer, etc, etc.

Ce type de collaboration directe est fondamental dans Scrum et en fait dans tout développement logiciel agile.

Que faire si le directeur de produit persiste à dire qu'il n'a pas le temps de participer aux réunions de planning de sprint ? Habituellement j'essaie les stratégies suivantes, dans l'ordre proposé :

- Essayer d'aider le directeur de produit à comprendre pourquoi sa participation directe est cruciale, et espérer qu'il change d'avis.
- Essayer de convaincre quelqu'un de l'équipe de se porter volontaire pour servir de représentant du directeur de produit pendant la réunion. Dire au directeur de produit : « Puisque que vous ne pouvez pas venir, nous laisserons Jeff ici présent vous représenter. Il aura les pleins pouvoirs pour changer la priorité et la portée des histoires pendant la réunion. Je vous suggère de vous synchroniser avec lui le mieux possible avant la réunion. Si Jeff ne vous convient pas comme représentant, merci de suggérer quelqu'un d'autre, à condition que cette personne puisse rester avec nous pour toute la durée de la réunion. »
- Essayer de convaincre l'équipe de direction de désigner un nouveau directeur de produit.
- Repousser le démarrage du sprint jusqu'à ce que le directeur de produit trouve le temps de participer à la réunion. En attendant, refuser de s'engager sur une quelconque livraison. Laisser l'équipe consacrer chaque jour à ce qui lui paraît le plus important ce jour-là.

De bonnes choses dans cette section ! <me tapotant sur le dos>
Juste une chose – je recommande fortement de séparer l'affinage de backlog (estimation, découpage des histoires, etc.) en une réunion séparée de manière à ce que le planning de sprint puisse être plus focalisé. La participation du product owner est toutefois toujours cruciale, dans les deux meetings.

Pourquoi la qualité n'est pas négociable

Dans le triangle ci-dessus j'ai intentionnellement évité une quatrième variable *qualité*.

- La *qualité externe* est ce qui est perçu par les utilisateurs du système. Une interface utilisateur lente et pas intuitive est un exemple de mauvaise qualité externe.
- La qualité interne fait référence à des points qui habituellement ne sont pas visibles par l'utilisateur, mais qui ont un profond effet sur la maintenabilité du système. Des choses comme la cohérence de la conception, la couverture de test, la lisibilité du code, les remaniements (*refactoring*).

En général, un système avec une qualité interne élevée peut tout de même avoir une qualité externe faible. Mais un système avec une faible qualité interne aura rarement une qualité externe élevée. Il est difficile de bâtir quelque chose de bien sur des fondations pourries.

Je traite la qualité externe comme faisant partie de la portée. Dans certains cas il peut y avoir d'excellentes raisons pour livrer une version du système qui aurait une interface utilisateur lente et peu élégante, et livrer ensuite une version nettoyée plus tard. Je laisse ce compromis au directeur de produit, puisqu'il est responsable de définir la portée.

Toutefois la qualité interne n'est pas sujette à discussion. C'est la responsabilité de l'équipe de maintenir la qualité du système dans toutes les circonstances et ceci est simplement non négociable. Jamais.

(Eh bien, OK, *presque* jamais)

Alors comment faisons-nous la différence entre les problèmes de qualité interne et ceux de qualité externe ?

Considérons que le directeur de produit dise : « OK les gars, je respecte votre estimation de temps de 6 points d'histoire, mais je suis certain que vous pouvez faire quelque chose de rapide pour ça en moitié moins de temps pour peu que vous y réfléchissiez. »

Aha ! Il est en train d'essayer d'utiliser la qualité interne comme variable. Comment je le sais ? Parce qu'il veut que nous réduisions l'estimation de l'histoire sans « payer le prix » de réduire la portée. Les mots « quelque chose de rapide » devraient déclencher une alarme dans votre tête...

Et pourquoi est-ce que nous n'autorisons pas ça ?

Mon expérience est que sacrifier la qualité interne est presque toujours une très très mauvaise idée. Le temps économisé est largement dépassé par le coût à court et à long terme. Une fois qu'une base de code est autorisée à commencer à se détériorer, il est très difficile d'y remettre de la qualité plus tard.

A la place, j'essaie de faire avancer la discussion vers la portée. « Puisqu'il est important pour vous d'avoir cette fonctionnalité rapidement, pouvons-nous réduire sa portée pour qu'elle soit plus rapide à implémenter ? Peut-être que nous pourrions simplifier la gestion d'erreur et avoir 'Gestion d'erreur avancée' comme histoire utilisateur séparée à conserver pour plus tard ? Ou bien pouvons-nous réduire la priorité des autres histoires pour pouvoir nous focaliser sur celle-ci ? »

Une fois que le directeur de produit a appris que la qualité interne n'était pas négociable, il devient normalement assez bon à manipuler les autres variables à la place.

En principe, oui. Mais en pratique aujourd'hui, j'ai tendance à être plus pragmatique. Parfois cela fait tout à fait sens en terme business de sacrifier la qualité à court terme – par exemple, parce que l'on a ce salon professionnel super important qui arrive après le prochain sprint ou parce que l'on a juste besoin d'un prototype pour valider une hypothèse au sujet d'un comportement utilisateur. Mais dans ces cas, le product owner a besoin de préciser pourquoi on le fait, et de s'engager à laisser l'équipe rembourser la dette technique dans le futur proche (parfois l'équipe va ajouter une histoire de « nettoyage » dans le backlog de produit, comme rappel). Une grande qualité interne devrait être la norme, et les exceptions devraient être traitées comme exceptionnelles.

Les réunions de planning de sprint qui s'éternisent...

La plus grande difficulté des réunions de planning de sprint est que :
1) les gens ne pensent pas qu'elles vont durer si longtemps
2) ... mais c'est toujours le cas !

Non, ça ne l'est pas ! Pas si vous faites l'affinage du backlog dans une réunion séparée. Beaucoup d'équipes que j'ai vues se retrouvent hebdomadairement pour une heure d'affinage de backlog, de manière à ce que le planning de sprint puisse se focaliser sur, et bien, le planning du sprint ! Cela donne également au product owner plus de chances de discuter et d'améliorer le backlog *avant* la réunion de planning de sprint, ce qui en retour raccourcit la réunion. Directive : une réunion de planning de sprint ne devrait normalement pas prendre plus d'une heure par semaine de sprint (considérablement moins pour les équipes expérimentées), alors trois heures ou moins pour des sprints de trois semaines.

Tout dans Scrum est en temps limité. J'adore cette règle simple et cohérente.
Nous essayons de nous y tenir.

Donc que faisons-nous quand la réunion de planning de sprint en temps limité est proche de sa fin et qu'il n'y a pas signe d'un objectif de sprint ou d'un backlog de sprint ? Est-ce que nous écourtons la réunion ? Ou nous continuons pour encore une heure ? Ou nous arrêtons la réunion et continuons le jour suivant ?

Cela arrive encore et toujours, en particulier pour les nouvelles équipes. Donc que faites-vous ? Je ne sais pas. Mais que faisons-nous? Oh, euh, eh bien, habituellement j'écourte brutalement la réunion. Point final. Laissons le sprint souffrir. Plus précisément, je dis à l'équipe et le directeur de produit « alors cette réunion se finit dans 10 minutes. Nous n'avons pas grand-chose comme plan de sprint en fait. Est-ce que nous faisons avec ce que nous avons, ou bien est-ce nous organisons une autre réunion de planning de sprint de 4 heures demain matin à 8h ? » Vous pouvez devinez ce qu'ils répondent… :o)

J'ai essayé de laisser la réunion s'éterniser. Habituellement ca ne sert à rien car les gens sont fatigués. S'ils n'ont pas produit un plan de sprint décent en 2-8 heures (ou autre limite de temps à votre convenance), ils n'y arriveront probablement pas en une heure de plus. L'autre option (organiser une nouvelle réunion le jour suivant) n'est pas mal non plus. Sauf que les gens sont généralement impatients de démarrer le sprint, et pas de passer encore quelques heures à planifier.

Donc j'écourte. Et en effet le sprint souffre. Le côté positif, toutefois, est que l'équipe a appris une leçon de forte valeur, et la réunion de planning de sprint *suivante* sera *bien* plus efficace. De plus les gens résisteront moins quand vous proposerez une durée de réunion qu'ils auraient auparavant considérée comme trop longue.

Apprenez à respecter vos limites de temps, apprenez à définir des limites réalistes. Ceci s'applique à la fois à la durée des réunions et des sprints.

Une fois, j'ai rencontré une équipe qui disait "On a essayé Scrum, on a détesté, plus jamais !" J'ai demandé pourquoi, et ils ont dit « Trop de temps passé en réunions ! On n'a jamais terminé quoi que ce soit. » J'ai demandé quelle réunion prenait le plus de temps, et ils ont dit le planning de sprint. J'ai demandé combien de temps cela prenait, et ils ont dit « Deux ou trois jours ! ». Deux ou trois JOURS de planning pour chaque sprint ?! Pas étonnant qu'ils aient détesté ! Ils ont loupé la règle du time-boxing : décidez en avance du temps que vous êtes enclin à investir, et puis tenez vous y ! Scrum est comme tout autre outil – vous pouvez utiliser un marteau pour construire quelque chose ou pour écraser votre pousse. Dans tous les cas, ne blâmez pas l'outil.

L'ordre du jour de la réunion de planning de sprint

Avoir une sorte d'ordre du jour préliminaire pour cette réunion réduira le risque de ne pas respecter la limite de temps.

Voici un exemple d'un plan typique pour nous.

Réunion de planning de sprint : 13:00 – 17:00 (10 minutes de pause toutes les heures)

- **13:00 – 13:30.** Le directeur de produit décrit le but du sprint et résume le backlog de produit. Le lieu, la date et l'heure de la démonstration sont fixés.
- **13:30 – 15:00**. L'équipe fait les estimations en temps, et décompose les éléments selon les besoins. Le directeur de produit met à jour les

niveaux d'importance selon les besoins. Les éléments sont clarifiés. « Comment démontrer » est explicité pour chacun des éléments les plus importants.

- **15:00 – 16:00**. L'équipe sélectionne les histoires à inclure dans le sprint. Elle fait les calculs de vélocité pour se confronter à la réalité.
- **16:00 – 17:00**. On choisit le lieu et l'heure pour la mêlée quotidienne (s'ils sont différents du dernier sprint). On poursuit la décomposition des histoires en tâches.

Ce bout entre 13:30 et 15:00, c'est l'affinage du backlog de produit (j'avais l'habitude de l'appeler « toilettage du backlog », puis j'ai appris que toilettage voulait dire de Mauvaises Choses dans certaines cultures). Mettez cela dans une réunion séparée et, voilà, vous avez des réunions de planning de sprint plus courtes et plus sympa. Ok, quelques ajustements mineurs peuvent être nécessaires, mais une grande partie de l'affinage du backlog devrait être faite *avant* le planning de sprint.

Ce plan n'est en aucune façon respecté trop strictement. Le Scrum master peut allonger ou raccourcir ces limites de temps comme nécessaire au fur et à mesure que la réunion progresse.

Le choix de la durée du sprint

L'un des produits de cette réunion de planning de sprint est une date de démonstration bien définie. Cela signifie que vous devez décider d'une durée de sprint.

Donc quelle la bonne durée d'un sprint ?

Eh bien, les sprints courts sont bénéfiques. Ils autorisent l'entreprise à être « agile », c'est-à-dire à changer souvent de direction. Les sprints courts = cycle de feedback court = des livraisons plus fréquentes = plus de feedback des clients = moins de temps passé à courir dans la mauvaise direction = apprendre et améliorer plus rapidement, etc.

Cela dit, les longs sprints sont bien aussi. L'équipe a plus temps pour monter en puissance, ils ont plus de temps pour récupérer des problèmes et parvenir tout de même à atteindre le but du sprint, il y a moins de surcharge en terme de réunions de planning de sprint, de démonstrations, etc.

En général les directeurs de produit aiment les sprints courts et les développeurs aiment les sprints longs. Donc la durée du sprint est un compromis. Nous avons beaucoup expérimenté à ce sujet et avons découvert notre longueur favorite : 3 semaines. La plupart de nos équipes (mais pas toutes) font des sprints de 3

semaines. C'est assez court pour nous donner suffisamment d'agilité d'entreprise, et suffisamment long pour permettre à l'équipe d'avoir du débit et de récupérer des problèmes qui surgissent au cours du sprint.

L'une de nos conclusions est : *faites* des expériences avec la durée des sprints au début. Ne perdez pas trop de temps *à analyser*, choisissez simplement une durée de sprint décente, essayez là pour un sprint ou deux, puis changez cette durée.

Toutefois, une fois que vous avez décidé quelle durée vous préférez, tenez-y vous pour un bon moment. Après quelques mois d'expérimentation, nous avons trouvé que 3 semaines c'était bien. Donc nous faisons des sprints de 3 semaines, point final. De temps en temps cela va paraître légèrement trop long, de temps en temps légèrement trop court. Mais en gardant toujours cette durée, cela devient comme un battement de cœur d'entreprise, dans lequel tout le monde s'installe confortablement. Il n'y a pas de débat au sujet de dates de livraison et autres parce que tout le monde sait que toutes les 3 semaines il y a une livraison, point final.

La plupart des équipes Scrum que je rencontre (presque toutes, en fait) finissent par effectuer des sprints de deux ou trois semaines. Une semaine est presque toujours trop court (« On a à peine commencé le sprint, et maintenant on planifie déjà la démo ! C'est stressant ! On n'arrive jamais à monter en vitesse et à profiter du flux de dévelopement ! »). Et quatre semaines est presque toujours trop long (« Nos réunions de planning sont une torture, et nos sprints n'arrêtent pas d'être interrompus ! »). Juste une observation.

La définition du but du sprint

Cela arrive presque à chaque fois. A un certain moment durant la réunion je demande « alors quel est le but de ce sprint ? » et tout le monde me regarde fixement et le directeur de produit fronce les sourcils et se gratte le menton.

Il se trouve qu'il est *difficile* de déterminer un objectif de sprint. Mais toutefois j'ai découvert que c'est vraiment payant de se forcer à en trouver un. Il vaut mieux un but à demi-merdique que pas de but du tout. Le but pourrait être « gagner plus d'argent » ou « terminer les trois histoires les plus prioritaires » ou « impressionner le PDG » ou « rendre le système suffisamment bon pour être déployé chez un groupe de beta-testeurs » ou « ajouter un support basique pour le back office » ou n'importe quoi d'autre. Le point important est que le but devrait être défini en termes métier, pas en termes techniques. Cela signifie en termes que les gens en dehors de l'équipe peuvent comprendre.

Le but du sprint devrait répondre à la question fondamentale « Pourquoi faisons-nous ce sprint ? Pourquoi est-ce que nous ne partons pas tous en vacances à la place ? » En fait un moyen d'extraire un objectif de sprint du directeur de produit est de poser littéralement cette question.

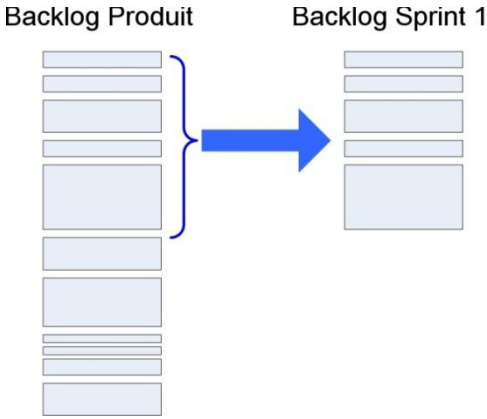
Le but devrait être quelque chose qui n'a pas déjà été réalisé. « Impressionner le PDG » peut être un bon but, mais pas s'il a déjà été impressionné par le système tel qu'il est maintenant. Dans ce cas tout le monde pourrait rentrer à la maison et le but du sprint serait quand même réalisé.

Le but du sprint peut sembler bête ou forcé durant le planning de sprint, mais il sert souvent à mi-sprint, quand les gens commencent à devenir perplexes sur ce qu'ils devraient faire. Si vous avez plusieurs équipes Scrum (comme nous) qui travaillent sur différents produits, il est très utile de lister les buts de sprint de toutes les équipes sur une seule page de wiki (ou équivalent) et de les mettre à un endroit bien visible, de telle sorte que tout le monde dans l'entreprise (pas seulement le management de haut niveau) sache ce que l'entreprise est en train de faire – et pourquoi !

OK, même le Guide Scrum est d'accord avec cela et dit que tous les sprints devraient avoir un objectif de sprint. Mais je trouve que ce n'est pas important d'avoir un objectif au niveau sprint ; cela peut être tout aussi bien d'avoir un objectif haut-niveau qui couvre plusieurs sprints, ou le prochain cycle de release. Assurez-vous juste qu'un sprint est quelque chose de plus important que juste « aller abattre un groupe d'histoires », ou vous risqueriez de trouver l'équipe dans un sérieux cas d'Ennui.

Le choix des histoires à inclure dans le sprint

L'une des activités principales de la réunion de planning de sprint est de décider quelles histoires inclure dans le sprint. Plus précisément, quelles histoires du backlog de produit il faut copier dans le backlog de sprint.



Regardez l'image ci-dessus. Chaque rectangle représente une histoire, triée par importance. L'histoire la plus importante est au sommet de la liste. La taille de chaque rectangle représente la taille de cette histoire (c'est-à-dire l'estimation de temps en points d'histoire). La hauteur de la parenthèse bleue représente la *vélocité estimée* de l'équipe, c'est-à-dire combien de points d'histoire l'équipe croit qu'elle pourra terminer durant le prochain sprint.

Le backlog de sprint à droite est un ensemble d'histoires du backlog de produit. Il représente la liste des histoires sur lesquelles l'équipe va s'engager durant ce sprint.

L'équipe décide combien d'histoires vont être prises dans le sprint. Pas le directeur de produit ou qui que ce soit d'autre.

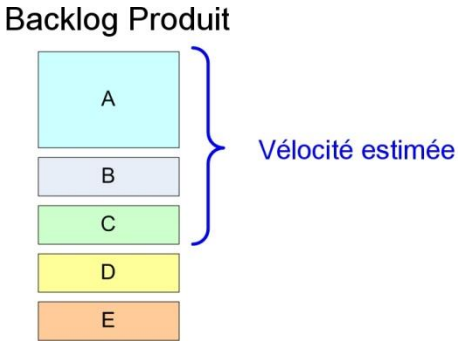
Cela soulève deux questions :

1. Comment l'équipe décide-t-elle quelles histoires inclure dans le sprint ?
2. Comment le directeur de produit peut-il influencer leur décision ?

Je vais commencer par la deuxième question.

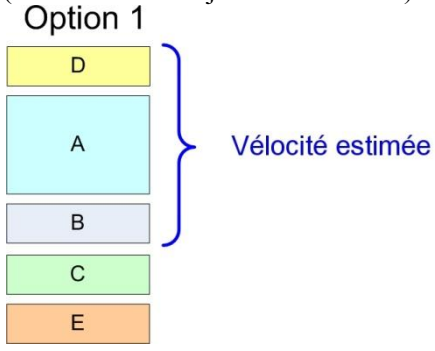
Comment le directeur de produit peut-il exercer une influence sur les histoires qui iront dans le sprint ?

Supposons que nous avons la situation suivante durant une réunion de planning de sprint.

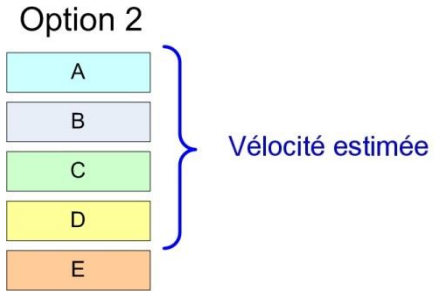


Le directeur de produit est déçu que l'histoire D ne soit pas incluse dans le sprint. Quelles sont ses options durant la réunion ?

Une option est de redéfinir les priorités. S'il donne à l'élément D le plus haut niveau d'importance, l'équipe sera obligée de l'ajouter en premier dans le sprint (et dans ce cas d'éjecter l'histoire C).

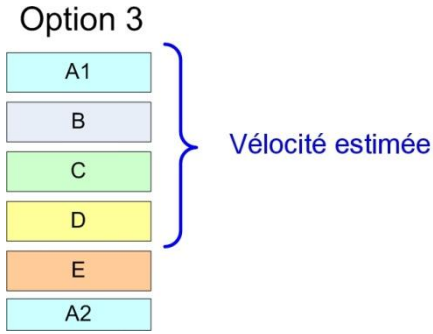


La deuxième option est de changer la portée – réduire la portée de l'histoire A jusqu'à ce que l'équipe croie que l'histoire D va tenir dans le sprint.



La troisième option est de partager une histoire. Le directeur de produit pourrait décider qu'il y a certains aspects de l'histoire A qui ne sont finalement pas si

importants, et donc il partage A en deux histoires A1 et A2 avec des niveaux d'importance différents.



Comme vous le voyez, bien que le directeur de produit ne puisse normalement pas contrôler la vélocité estimée, il y a plusieurs moyens par lesquels il peut exercer une influence sur les histoires qui seront prises dans le sprint.

Comment les équipes choisissent-elles les histoires à inclure dans le sprint?

Pour cela, nous utilisons deux techniques :

1. L'intuition
2. Les calculs de vélocité

Estimation en utilisant l'intuition

- **Scrum master**: «Eh les gars, peut-on finir l'histoire A dans ce sprint?» (montrant l'élément le plus important du backlog de produit)
- **Lisa**: «Ben. Bien sûr que nous pouvons. Nous avons 3 semaines, et c'est une fonctionnalité plutôt facile.»
- **Scrum master**: «OK, et nous rajoutons aussi l'histoire B ?» (montrons le deuxième élément le plus important)
- **Tom & Lisa ensemble**: «Pas de problème.»
- **Scrum master**: «OK, et avec les histories A, B et C maintenant ?»
- **Sam (à l'adresse du Directeur de produit)**: «Est-ce que l'histoire C comporte une gestion avancée des erreurs ?»

- **Directeur de produit**: «non, vous pouvez la laisser de coté pour le moment, implémentez juste une gestion basique des erreurs. »

- **Sam**: «alors C devrait être bonne aussi.»

- **Scrum master**: «OK, et si nous ajoutons l'histoire D ?»

- **Lisa**: «Hum....»

- **Tom**: «Je pense que nous pouvons le faire.»

- **Scrum master**: «Confiant à 90% ? à 50% ?»
- **Lisa & Tom**: «A peu près 90%. »

- **Scrum master**: «OK, prenons D. Et si nous ajoutons l'histoire E ?»

- **Sam**: «Peut-être.»

- **Scrum master**: «90%? 50%?»

- **Sam**: «Je dirais plus proche de 50%. »

- **Lisa**: «Je ne suis pas sûre.»

- **Scrum master**: «OK, alors ne la prenons pas. Nous nous engagerons pour A, B, C, et D. Nous finirons E bien sûr si nous le pouvons, mais personne ne devrait compter dessus, du coup nous la laisserons en dehors du plan du sprint. Qu'en dîtes-vous?»

- **Tout le monde**: «OK!»

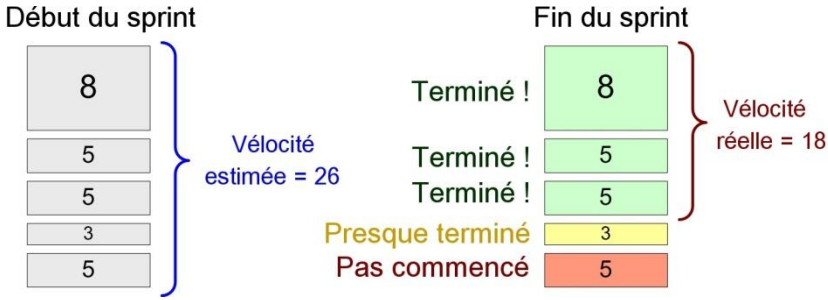
L'intuition marche plutôt bien pour des petites équipes et des sprints courts.

Estimation en utilisant les calculs de vélocité

Cette technique implique deux étapes:
1. Décider la *vélocité estimée*
2. Calculer combien d'histoires vous pouvez ajouter sans dépasser la vélocité estimée

La vélocité mesure la «quantité de travail fini», où chaque élément est pondéré selon son estimation initiale.

L'image ci-dessous montre un exemple de *vélocité estimée* au démarrage d'un sprint et la *vélocité effective* à la fin du sprint. Chaque rectangle est une histoire, et le numéro à l'intérieur est l'estimation initiale de cette histoire.



Notez que la vélocité effective est basée sur les estimations *initiales* de chaque histoire. Les mises à jour des estimations du temps de l'histoire faites durant le sprint sont ignorées.

Déjà, j'entends votre objection : « En quoi est-ce utile ? Une vélocité basse ou élevée peut dépendre de tout un tas de facteurs ! Des programmeurs stupides, des estimations initiales incorrectes, glissement de périmètre, perturbations inattendues durant le sprint, etc. ! »

Je suis d'accord, il s'agit d'un nombre approximatif. Mais cela reste un nombre utile, spécialement quand il est comparé à rien du tout. Il vous donne de solides faits. «Quelles que soient les raisons, voici la différence entre combien nous avions pensé pouvoir faire et combien nous avons réellement pu finir».

Que dire d'une histoire qui est *presque* finie durant un sprint ? Pourquoi ne pas compter les points partiellement accomplis pour celle-ci dans notre vélocité effective ? Eh bien c'est pour souligner le fait que Scrum (en fait le développement logiciel agile et le lean manufacturing en général) est entièrement focalisé sur l'obtention de quelque chose de complètement terminé, pouvant être livré ! La valeur d'une chose à moitié fini est zéro (peut-être en fait même négative). Lisez «Managing the Design Factory» de Donald Reinertsen ou un des livres des Poppendieck pour en savoir plus sur le sujet.

Alors à travers quel arcane magique estimons-nous la vélocité ?

Une façon très simple d'estimer la vélocité est d'étudier le passé de l'équipe. Quelle a été leur vélocité durant les sprints passés ? Alors on fait l'hypothèse que la vélocité sera grosso modo la même au prochain sprint.

Cette technique est connue sous le nom de la *météo de la veille*. Cela est faisable seulement avec des équipes qui ont déjà fait quelques sprints (des statistiques sont alors disponibles) et qui feront le prochain sprint de la même façon, avec la même taille d'équipe et dans les mêmes conditions de travail, etc. Bien sûr ceci n'est pas toujours le cas.

Une variante plus sophistiquée consiste à faire un simple calcul de ressource. Disons que nous planifions un sprint de 3 semaines (15 jours de travail) avec une équipe de 4 personnes. Lisa est en congés durant 2 jours. Dave est disponible seulement à 50% et il sera en congés durant 1 jour. Mettant tout cela ensemble…

…cela nous donnes 50 jours-homme pour ce sprint.

Attention: voici une partie que je déteste vraiment. J'aimerais déchirer les quelques pages qui suivent ! Mais allez-y et lisez-les si vous êtes curieux, et je vous expliquerai pourquoi après.

Est-ce cela notre vélocité estimée ? Non ! Car notre unité d'estimation est le point d'histoire qui, dans notre cas, correspond à peu près à autant de «*jours-homme idéaux*». Un jour-homme idéal est un jour de travail, parfaitement efficace, sans perturbation, ce qui est rare. De plus, nous devons tenir compte de choses comme le travail inattendu qui va s'ajouter au sprint, des personnes malades, etc.
Du coup, notre vélocité estimée sera certainement inférieure à 50. Mais de combien inférieure ? Nous utilisons à cette fin le terme «facteur de focalisation».

Le facteur de focalisation est une estimation indiquant à quel point l'équipe est concentrée. Un faible facteur de focalisation devrait signifier que l'équipe s'attend à avoir de nombreuses perturbations ou s'attend à ce que ses estimations soient trop optimistes.

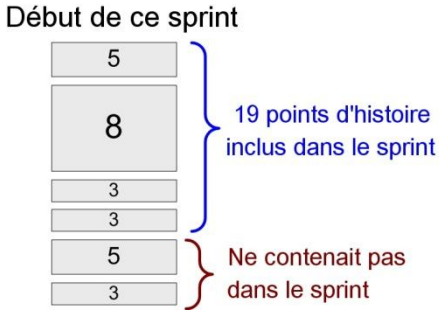
Bla bla bla. Du charabia de maths. Utilisez juste la météo de la veille (ou votre intuition si vous n'avez pas de données) et ignorez cette bêtise de facteur de focalisation.

La meilleure façon de déterminer un facteur de focalisation raisonnable est de regarder le dernier sprint (ou même mieux, la moyenne des quelques derniers sprints).

La *vélocité* effective est la somme des estimations initiales de toutes les histories qui ont été terminées durant le dernier sprint.

Disons que le dernier sprint a fait 18 points d'histoire avec une équipe de 3 personnes constituée de Tom, Lisa et Sam travaillant durant 3 semaines pour un total de 45 jours-homme. Et maintenant nous essayons d'avoir une idée de notre vélocité estimée pour le sprint qui arrive. Pour compliquer les choses, un nouveau gars Dave rejoint l'équipe pour ce sprint. Prenant en compte les congés et le reste nous obtenons 50 jours-homme pour le prochain sprint.

Notre vélocité estimée pour le prochain sprint est donc de 20 points d'histoire. Cela veut dire que l'équipe devrait rajouter des histoires au sprint jusqu'à cela atteigne environ 20.



Dans ce cas l'équipe peut choisir les 4 histoires en tête pour un total de 19 points d'histoire, ou les 5 histoires en tête pour un total de 24 points. Disons qu'ils choisissent 4 histoires, car cela est le plus proche de 20 points. En cas de doute, choisissez moins d'histoires.

Comme ces 4 histoires font au total 19 points d'histoires, leur vélocité estimée finale pour ce sprint est 19.

La météo de la veille est une technique pratique mais utilisez la avec une dose de bon sens. Si le dernier sprint était anormalement mauvais car la plupart des membres de l'équipe était malade pendant une semaine, alors il serait prudent de considérer que vous ne serez pas malchanceux une seconde fois et vous pourrez estimer un facteur de focalisation plus élevé pour le prochain sprint. Si l'équipe a mis en place récemment un nouveau système d'intégration continu aussi rapide que la foudre, alors vous pouvez probablement augmenter le facteur de focalisation en raison de cela. Si une nouvelle personne va rejoindre l'équipe, vous devez baisser le facteur de focalisation pour prendre en compte la formation de ce dernier, etc.

Chaque fois que cela est possible, regardez en arrière de plusieurs sprints et faites la moyenne des chiffres afin d'obtenir des estimations plus sûres.

Que faire si l'équipe est complètement nouvelle et que vous n'avez aucune statistique? Observez le facteur de focalisation d'autres équipes sous des conditions similaires.

Que faire si vous n'avez pas s'autre équipe à observer? Devinez le facteur de focalisation. La bonne nouvelle est que votre intuition ne jouera que pour le premier sprint. Après cela vous aurez des statistiques, vous pourrez mesurer de façon continue et améliorer votre facteur de focalisation et la vélocité estimée.

Le facteur de focalisation que j'utilise par défaut pour les nouvelles équipes est généralement 70%, car c'est là que la plupart de nos équipes ont abouti au fil du temps.

Quelle technique d'estimation utilisons-nous?

J'ai mentionné plusieurs techniques au dessus – l'intuition, le calcul de vélocité basé sur la météo de la veille, et le calcul de vélocité basé sur le nombre de jours-homme disponible et l'estimation du facteur de focalisation.

Alors quelle technique utilisons-nous?

Nous combinons généralement toutes ces techniques à un certain degré. Cela ne prend pas longtemps.

Nous regardons le facteur de focalisation et la vélocité effective du dernier sprint. Nous regardons le total de nos ressources disponibles pour ce sprint et estimons le facteur de focalisation. Nous discutons de la moindre différence entre ces deux facteurs de concentration et procédons à des ajustements si nécessaire.

OK, c'est la fin de la section douloureuse. Je n'ai plus jamais utilisé le facteur de focalisation parce que cela prend du temps, donne une fausse impression de précision, et vous force à estimer les histoires en jours-homme idéaux.

Aussi, le facteur de focalisation porte l'hypothèse que plus de personnes = plus grande vélocité. Parfois c'est vrai, mais parfois pas. Si l'on ajoute une nouvelle personne dans l'équipe, la vélocité va généralement *baisser* les deux premiers sprints, vu que l'équipe passe du temps pour faire monter la nouvelle personne. Si une équipe est trop grande (comme plus de 10 personnes), la vélocité va sans aucun doute baisser. Aussi, le terme « facteur de focalisation » implique qu'une valeur de moins de 100% signifie que l'équipe n'est pas focalisée, ce qui envoie un message trompeur au management.

Alors sautez tous ces facteurs de focalisation et jours-homme. Regardez juste ce que vous avez terminé lors des derniers sprints, en comptant les points d'histoire, ou même juste en comptant le nombre d'histoires si vous n'avez pas d'estimations du tout. Puis prenez grossièrement le même nombre d'histoires pour ce sprint. Si vous avez des perturbations planifiées dans le sprint (comme deux personnes absentes pour une formation) alors enlevez quelques histoires jusqu'à ce que cela vous semble correct. Le moins de données historiques vous avez, le plus vous avez besoin de vous fier à votre intuition.

Faites cela et vos réunions de planning de sprint vont être plus courtes, plus efficaces et plus amusantes. Et, contre-intuitivement, vos plans vont probablement finir par être plus précis.

Une fois que nous avons une liste préliminaire des histoires à inclure dans le sprint, je fais une vérification à «l'intuition». Je demande à l'équipe d'ignorer les chiffres pour un moment et de *sentir* tout simplement si cela semble être un morceau raisonnable à se mettre sous la dent pour un sprint. Si cela semble trop gros, nous enlevons une histoire ou deux. Et vice-versa.

A la fin de la journée, l'objectif est simplement de décider des histoires à inclure dans le sprint. Le facteur de focalisation, la disponibilité des ressources, et la vélocité estimée sont juste des moyens pour atteindre ce but.

Pourquoi nous utilisons des fiches

La plupart des réunions de planification de sprint se déroulent en se basant sur les histoires du backlog de produit. On les estime, on redéfinit leur priorités, on les clarifie, on les découpe, etc..

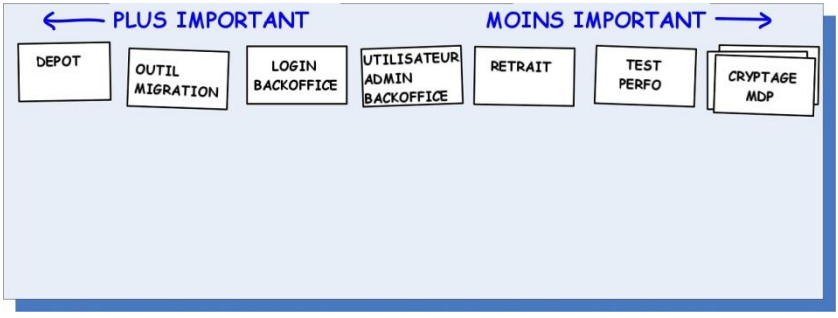
Comment faisons-nous ça en pratique ?

Eh bien, par défaut, les équipes avaient l'habitude d'allumer le projecteur vidéo, de montrer le backlog sous Excel, et un gars (typiquement le Directeur de produit ou le Scrum master) prenait le clavier, marmonnait en parcourant chaque histoire et invitait à discuter. Comme l'équipe et le Directeur de produit discutaient les priorités et les détails le gars au clavier mettait à jour directement l'histoire dans Excel.

Cela paraît bien, non ? En fait, ça ne l'est pas. Généralement ça craint. Et le pire c'est qu'en général l'équipe ne *remarque* pas que ça craint jusqu'à ce qu'arrive la fin de la réunion et réalise qu'ils n'ont toujours pas réussi à parcourir toute la liste des histoires !

Oh, la souffrance….

Une solution qui marche beaucoup mieux consiste à créer des *fiches* et *de les mettre au mur* (ou sur une grande table).

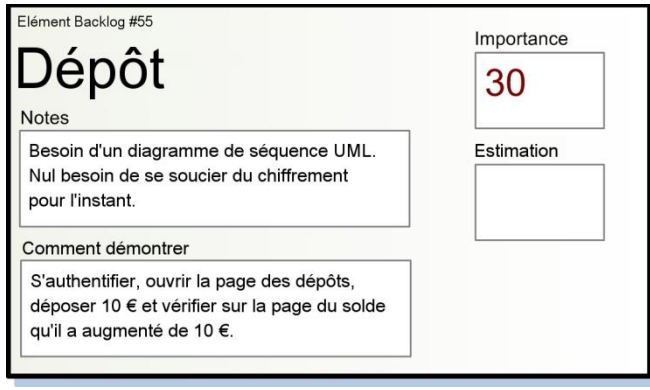


Il s'agit d'une interface utilisateur supérieure comparée au projecteur, car:

- Tout le monde se sent plus impliqué personnellement (plutôt que juste le gars avec le clavier)
- Les gens sont debout et marchent autour => ils restent éveillés, plus en alerte
- Plusieurs histoires peuvent être modifiées en même temps
- Changer la priorité est triviale – il faut juste déplacer les fiches

- Après la réunion, les fiches peuvent être transférées directement vers la salle de l'équipe et être utilisées comme panneau mural des tâches (voir page 45 « Comment nous faisons les backlogs de sprint)

Vous pouvez soit les écrire à la main ou (comme nous le faisons habituellement) utiliser un simple script qui génère des fiches à imprimer directement à partir du backlog de produit.



PS – le script est disponible sur mon blog ici http://blog.crisp.se/henrikkniberg.

La manière la plus simple de trouver le script est de chercher « index card generator » sur google. Je ne peux pas croire que ce vieux hack est toujours d'actualité ! Des personnes bienveillantes ont aidé à le porter sur Google Spreadsheets également. Tout outil décent de gestion de backlog aura une fonction d'impression comme cela. Expérimentez avec différents outils et trouvez ce qui marche le mieux dans votre contexte. Assurez-vous juste que vous adaptez l'outil à votre processus, et pas vice versa.

Important: Après la réunion de planification de sprint, notre Scrum master met à jour le backlog de produit sous Excel, en respectant le moindre changement apporté physiquement sur les fiches. Oui, il s'agit d'une légère paperasserie administrative mais nous trouvons qu'elle est parfaitement acceptable en considérant à quel point la réunion de planification de sprint est efficace avec les fiches.

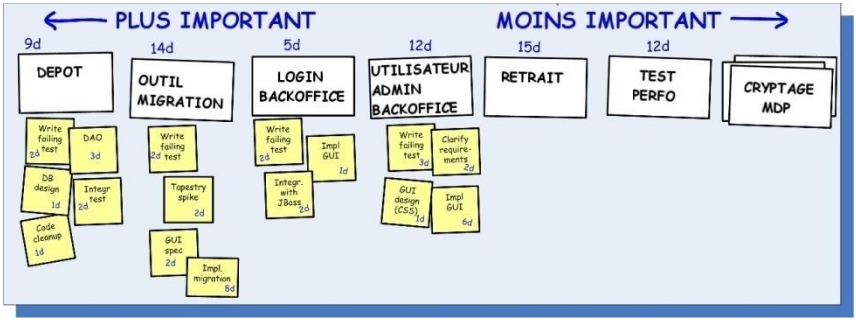
Une remarque à propos du champ «Importance». Il s'agit de l'importance spécifiée dans le backlog du produit sous Excel au moment de l'impression. L'avoir sur la fiche rend facile le tri des fiches par ordre d'importance (en général nous mettons les éléments les plus importants à gauche, et les moins importants à droite). Cependant, une fois que les fiches sont au mur vous pouvez ignorer le niveau d'importance et utiliser à la place l'ordre dans lequel les fiches

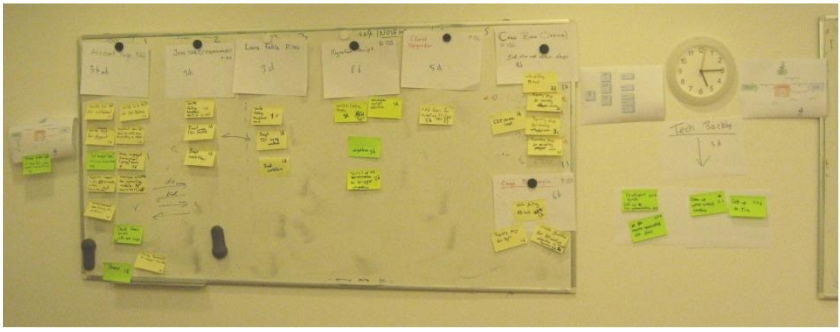
sont physiquement placées. Si le Directeur de produit permute deux éléments ne perdez pas de temps à mettre à jour le niveau d'importance sur le papier. Veillez seulement à ce que le niveau d'importance soit mis à jour dans le backlog du produit dans Excel à l'issue de la réunion.

Ou sautez juste l'évaluation d'importance. Oups, je l'ai déjà dit à plusieurs reprises. Est-ce que je me répète ? Est-ce que je me répète ?

Les estimations en temps sont généralement plus facile à faire (et plus précis) si une histoire est découpée en tâches. En fait nous utilisons le terme «activité» car le mot «tâches» veut dire quelque chose de *complètement* différent en suédois :o) Cela est aussi agréable et facile à faire avec nos fiches. Vous pouvez avoir l'équipe qui se divise en binômes et qui découpe une histoire chacun, en parallèle.

Physiquement, nous faisons cela en ajoutant de petits post-its sous chaque histoire, chaque post-it correspondant à une tâche de cette historie.





Nous ne mettons pas à jour le backlog du produit dans Excel avec notre découpage en tâche, pour deux raisons :

- Le découpage en tâches est généralement tout à fait éphémère, i.e. les tâches sont fréquemment modifiées et raffinées durant le sprint, de telle sorte qu'il serait trop harassant de garder le backlog du produit synchronisé dans Excel.

- Le Directeur de produit n'a pas besoin d'être impliqué à ce niveau de détail de toute façon.

De la même façon que les fiches, les post-its des tâches peuvent être directement réutilisées dans le backlog de sprint (voir page 45 «Comment nous faisons les backlogs de sprint»)

Définition de «terminé»

Il est important que le Directeur de produit et l'équipe s'accorde sur une définition claire de «terminé».

TRES important !

Une histoire est-elle terminée lorsque tout le code a été archivé ? Ou est-elle terminée seulement quand elle a été déployée sur un environnement de test et vérifiée par une équipe de test d'intégration ? Chaque fois que cela est possible, nous utilisons la définition de terminé «prêt à déployer en production» mais dès fois nous devons nous rabattre sur la définition de terminé «déployé sur serveur de test et prêt pour le test d'acceptation».

Au commencement nous utilisions des checklists détaillées pour cela. Maintenant souvent nous disons juste «une histoire est terminée quand le testeur de l'équipe Scrum le dit». C'est alors au testeur de s'assurer que l'intention du Directeur de produit est bien compris par l'équipe, et que l'élément est assez «terminé» pour satisfaire la définition requise de terminé.

OK, c'est un peu bancal. Une checklist concrète est plus utile – assurez-vous juste qu'elle n'est pas trop longue. Utilisez-la par défaut, pas comme une écriture sainte. Focalisez-vous sur les choses que les personnes ont tendance à oublier (comme « mettre à jour les notes de release » ou « pas d'ajout de dette technique » ou « obtenir un vrai feedback utilisateur »).

Nous sommes venus à réaliser que toutes les histoires ne peuvent pas être traitées de la même façon. Une histoire appelée «Formulaire de requête utilisateurs» sera traitée très différemment d'une histoire nommée «Guide des opérations». Dans le dernier cas, la définition de «terminé» pourrait simplement être «accepté par l'équipe des opérations».C'est pourquoi le bon sens est souvent meilleur que les checklists formelles.

Si la définition de terminé vous semble souvent confuse (ce qui était notre cas au début) vous devriez probablement avoir un champ «définition de terminé» pour chaque histoire individuelle.

L'estimation de temps avec le poker de planning

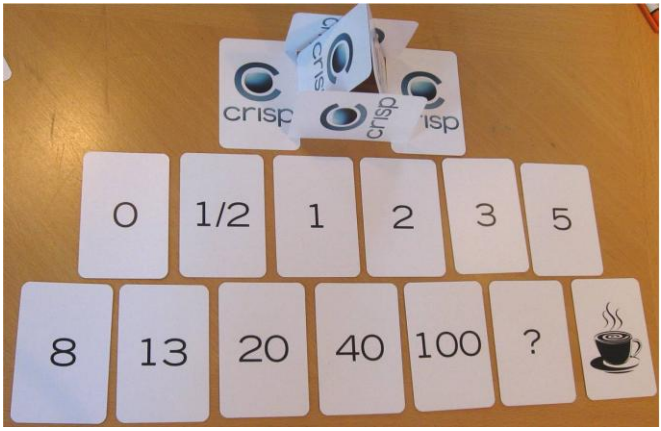
L'estimation est une activité d'équipe – chaque membre de l'équipe est habituellement impliqué dans l'estimation de chaque histoire. Pourquoi ?

- Au moment du planning, normalement nous ne savons pas exactement qui va implémenter quelle partie de quelle histoire.
- Les histoires impliquent normalement plusieurs personnes et plusieurs sortes d'expertise (conception d'interface utilisateur, codage, test, etc.).
- Pour pouvoir fournir une estimation, un membre d'équipe a besoin d'une certaine compréhension de l'histoire. En demandant à tout le monde d'estimer chaque élément, nous assurons que chaque membre d'équipe comprend chacun des éléments. Cela augmente la probabilité que les membres de l'équipe vont s'aider mutuellement durant le sprint. Cela augmente également la probabilité que des questions importantes sur l'histoire sont posées tôt.
- En demandant à tout le monde d'estimer une histoire nous découvrons souvent des situations où deux membres de l'équipe ont des estimations extrêmement différentes pour la même histoire. C'est bien mieux de découvrir et discuter ce genre de choses le plus tôt possible.

Si vous demandez à l'équipe de fournir une estimation, normalement la personne qui comprend le mieux l'histoire va se lancer en premier. Malheureusement cela influence fortement les estimations de tous les autres.

Il y a une excellente technique pour éviter cela – elle est appelée poker de planning (terme inventé par Mike Cohn je pense).

En fait, Mike dit qu'il l'a appris de James Grenning, et James va probablement dire qu'il a eu l'idée de quelqu'un d'autre. Peu importe. Nous nous tenons tous sur les épaules de géants. Ou peut-être que nous sommes une bande de nains nous tenant sur les épaules des uns et des autres. Euh, bref… – vous voyez ce que je veux dire.



Chaque membre de l'équipe reçoit un jeu de 13 cartes comme montré ci-dessus.

<pitch> Nous vendons ces jeux sur planningpoker.crisp.se. Et ils sont beaucoup plus cool maintenant que sur la photo, bien que vous pouvez probablement trouver moins cher si vous cherchez sur google. Oh, et sur ce site nous vendons aussi quelque chose de vraiment sympa qui s'appelle les Cartes de Jimmy, par mon collègue Jimmy (oui, on a eu du mal à trouver un nom pour les cartes). Jetez-y un œil ! </pitch>

Chaque fois qu'une histoire doit être estimée, chaque membre de l'équipe sélectionne une carte qui représente son estimation de temps (en points d'histoire) et la place sur la table face cachée. Quand tous les membres de l'équipe ont terminé, toutes les cartes sur la table sont révélées simultanément. De cette manière chaque membre de l'équipe est forcé à réfléchir par lui-même au lieu de s'appuyer sur l'estimation de quelqu'un d'autre.

S'il y a un gros écart entre deux estimations, l'équipe discute les différences et tente d'élaborer une vision commune du travail impliqué par l'histoire. Ils peuvent même faire une sorte de décomposition en tâches. Après, l'équipe estime de nouveau. Cette boucle est répétée jusqu'à ce que les estimations convergent, c'est-à-dire que toutes les estimations soient *approximativement* les mêmes pour cette histoire.

Il est important de rappeler aux membres de l'équipe qu'ils sont là pour estimer la quantité de travail total pour cette histoire. Pas seulement « leur » part du travail. Le testeur ne devrait pas estimer uniquement la quantité de travail de test.

Notez que la suite des nombres n'est pas linéaire. Par exemple il n'y a rien entre 40 et 100. Pourquoi ?

C'est pour éviter un faux sentiment de précision pour les grandes estimations de temps. Si une histoire est estimée a environ 20 points d'histoire, il n'est pas pertinent de discuter si ce devrait être 20 ou 18 ou 21. Tout ce que nous savons, c'est que c'est une grosse histoire et qu'elle est difficile à estimer. Donc 20 est notre estimation à la louche.

Vous voulez des estimations plus détaillées ? Découpez l'histoire en histoires plus petites et estimez plutôt les petites histoires !

Et non, vous ne pouvez pas tricher en combinant 5 et 2 pour faire un 7. Vous devez choisir soit 5 soit 8, il n'y a pas de 7.

Quelques cartes particulières :

- 0 = « cette histoire est déjà faite » or « cette histoire c'est pratiquement rien, juste quelques minutes de travail ».
- ? = « Je n'ai aucune idée. Vraiment aucune. »
- Tasse de café = « Je suis trop fatigué pour penser. Faisons une courte pause. »

Une autre société est arrivée avec un jeu encore plus cool, les cartes d'estimation No Bullshit (estimation.lunarlogic.io). Il n'a que trois cartes :

- 1 (one)
- TPG (trop p***** de gros)
- API (aucune p***** d'idée)

Plutôt sympa ! J'aurais bien voulu avoir cette idée. Je m'attribue toutefois le mérite pour les tasses de café.

La clarification des histoires

Le pire c'est quand un membre de l'équipe démontre fièrement une nouvelle fonctionnalité à la démonstration du sprint, et que le directeur de produit fronce les sourcils et dit « eh bien c'est sympa, mais ce n'est *pas ce que j'ai demandé !* »

Comment vous assurez vous que la compréhension d'une histoire par le directeur de produit corresponde à la compréhension de l'équipe ? Ou que chaque membre de l'équipe a la même compréhension de chaque histoire ? Eh bien vous ne pouvez pas. Mais il y a quelques techniques simples pour identifier les incompréhensions les plus flagrantes. La plus simple des techniques est de s'assurer que tous les champs sont bien remplis pour chaque histoire (ou plus précisément, pour chaque histoire qui a suffisamment d'importance pour être considérée pour ce sprint).

Certains appellent cela la « définition de prêt ». Alors la « définition de terminé » est une checklist pour quand une histoire est terminée, et la « définition de prêt » est une checklist pour quand une histoire est prête à être tirée dans un sprint. Très utile.

Exemple 1:

L'équipe et le directeur de produit sont satisfaits du plan du sprint et sont prêts à terminer la réunion. Le Scrum master dit « attendez une seconde, cette histoire appelée 'Ajouter un utilisateur', elle n'a pas d'estimation. Estimons-là ! » Après quelques tours de poker de planning, l'équipe s'accorde sur 20 points tandis que le directeur de produit éclate de colère « Qu'est-ce que ca veut dire ? ». Après quelques minutes de discussion animée, il s'avère que l'équipe n'avait pas compris la portée de 'Ajouter un utilisateur', ils pensaient que cela signifiait « une belle interface web pour ajouter, supprimer, et chercher des utilisateurs », tandis que le directeur de produit voulait juste dire « ajouter des utilisateurs en faisant manuellement du SQL sur la BD ». Ils estiment à nouveau et tombent sur 5 points.

Exemple 2:

L'équipe et le directeur de produit sont satisfaits du plan du sprint et sont prêts à terminer la réunion. Le Scrum master dit « attendez une seconde, cette histoire appelée 'Ajouter un utilisateur', comment faudrait-il la démontrer ? ». Quelques grommellements s'ensuivent, et après une minute quelqu'un se lève et dit « eh bien, d'abord on se connecte au site web, et puis … » mais le directeur de produit l'interrompt en disant « se connecter au site web ?! Non, non et non, cette fonctionnalité ne devrait pas du tout concerner le site web, ce devrait être un bête petit morceau de script SQL pour les administrateurs ».

Le « Comment démontrer » d'une histoire peut (et devrait) être *très court*. Sinon vous ne finirez pas le planning du sprint à temps. Essentiellement c'est une description de haut niveau en français de comment exécuter manuellement le scénario de test le plus typique. « Faire ceci, puis cela, et alors vérifier ceci ».

J'ai remarqué que cette description simple fait souvent apparaître des incompréhensions importantes sur la portée d'une histoire. C'est bien de les découvrir très tôt, n'est-ce pas ?

J'aime toujours cette technique et je l'utilise dès qu'une histoire semble vague. Cela rend les choses concrètes. Une alternative est de dessiner une esquisse de structure ou une liste de critères d'acceptation. Pensez à l'histoire comme l'expression d'un problème de haut niveau, et la « définition de terminé »

comme un exemple concret de à quoi cela pourrait ressembler lorsque ce sera terminé.

La décomposition des histoires en plus petites histoires

Les histoires ne devraient pas être trop petites ou trop grosses (en termes d'estimations). Si vous avez un lot d'histoires à 0,5 points vous allez probablement être une victime du micro-management. D'un autre côté, une histoire de 40 points implique un risque élevé de finir *partiellement* terminée, ce qui ne produit aucune valeur pour votre entreprise et augmente simplement le travail administratif. De plus, si votre vélocité estimée est de 70 et que vos deux histoires prioritaires sont à 40 points chacune, le planning devient quelque peu difficile. Vous devez choisir entre le sous-engagement (c'est-à-dire prendre juste un élément) et le sur-engagement (c'est-à-dire prendre les deux éléments).

Je remarque qu'il est presque toujours possible de couper une grosse histoire en plusieurs histoires plus petites. Faites simplement attention que les petites histoires continuent à représenter des livrables avec une valeur pour le client.

Normalement nous nous efforçons d'avoir des histoires pesant de 2 à 8 jours-hommes. Notre vélocité est habituellement autour de 40-60 pour une équipe typique, ce qui nous donne quelque chose comme environ 10 histoires par sprint. Quelquefois cela descend à 5 et quelquefois cela monte à 15. Cela reste un nombre de fiches cartonnées gérable.

De 5 à 15 histoires dans un sprint est un conseil utile. Moins de 5 signifie généralement que les histoires sont trop grosses pour la taille du sprint, alors que plus de 15 signifie généralement que l'équipe a trop tiré et ne finira pas tout (ou les histoires sont trop petites, causant du micro management).

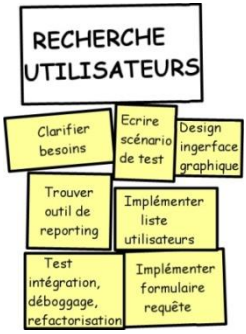
La décomposition des histoires en tâches

Attendez une seconde, quelle est la différence entre « tâches » et « histoire » ? C'est une très bonne question.

La distinction est assez simple. Les histoires sont des choses livrables qui intéressent le directeur de produit. Les tâches sont des choses non livrables, ou des choses auxquelles le directeur de produit ne s'intéresse pas.

Exemple de décomposition d'une histoire en histoires plus petites :

Exemple de décomposition d'une histoire en tâches :



Voici quelques observations intéressantes :

- Les nouvelles équipes Scrum rechignent à passer du temps à découper à l'avance un lot d'histoires en tâches comme cela. Certains ressentent cela comme une approche de type cycle en V.
- Pour les histoires bien comprises, c'est tout aussi simple de faire ce découpage à l'avance que de le faire plus tard.
- Ce type de découpage révèle souvent du travail supplémentaire qui entraîne une hausse de l'estimation, ce qui par conséquent donne un plan de sprint plus réaliste.
- Ce type de découpage à l'avance rend les mêlées quotidiennes visiblement plus efficaces (voir page 74 « Comment nous faisons les mêlées quotidiennes »).
- Même si le découpage est imprécis et changera une fois le travail démarré, les avantages ci-dessus restent valables.

Donc nous essayons d'avoir une limite de temps pour la réunion de planning de sprint suffisamment importante pour faire tenir tout cela, mais si nous manquons de temps nous laissons tomber (voir « Où tracer la limite » ci-dessous).

Le découpage en tâches est une bonne opportunité pour identifier les dépendances – comme « nous aurons besoin d'avoir accès aux logs de production » ou « nous aurons besoin de Jim en RH » – et déterminer des façons de gérer ces dépendances. Peut-être appeler Jim et voir s'il peut réserver du temps pour nous pendant le sprint. Le plus tôt vous découvrirez une dépendance, le moins de chances vous aurez que cela vous explose le sprint !

Note – nous pratiquons le DDT (développement dirigé par les tests) ce qui en pratique signifie que la première tâche pour presque toutes les histoires est « écrire un test qui échoue » et la dernière tâche est « remaniement (*refactor*) » (= améliorer la lisibilité du code et supprimer les duplications).

Le choix du lieu et l'heure pour la mêlée quotidienne

Un élément de sortie souvent oublié de la réunion de planning du sprint est « un lieu et une heure bien définis pour la mêlée quotidienne ». Sans cet élément votre sprint va faire un mauvais départ. La première mêlée quotidienne est essentiellement le point où tout le monde décide où commencer à travailler.

Je préfère les réunions du matin. Mais, je dois l'admettre, nous n'avons pas réellement essayé de faire les mêlées quotidiennes l'après-midi ou à mi-journée.

Maintenant je l'ai fait. Cela fonctionne bien. Laissez l'équipe décider. Si elle n'est pas sûre, expérimentez. La plupart des équipes préfèrent toutefois les matins.

Inconvénient des mêlées de l'après-midi : quand vous venez travailler le matin, vous devez essayer de vous rappeler ce que vous avez dit hier à vos collègues au sujet de ce que vous feriez aujourd'hui.

Inconvénient des mêlées du matin : quand vous venez travailler le matin, vous devez essayer de vous rappeler ce que vous avez fait hier pour pouvoir le rapporter.

Mon point de vue est que le premier inconvénient est pire, puisque la chose la plus importante est ce que vous *allez faire*, pas ce que vous *avez fait*.

Notre procédure par défaut est de choisir le moment le plus tôt où personne dans l'équipe ne se plaigne. Habituellement 9:00, 9:30, ou 10:00. La chose la plus importante est une heure que tout le monde dans l'équipe accepte de bon cœur.

Où tracer la limite ?

OK, le temps commence à manquer. Parmi toutes les choses que nous voulons faire durant le planning de sprint, qu'est-ce nous laissons de côté si le temps manque ?

Eh bien, j'utilise la liste de priorités suivante :

Priorité 1 : Un objectif de sprint et une date de démonstration. C'est vraiment le minimum dont vous avez besoin pour démarrer un sprint. L'équipe a un but, une date de fin et ils peuvent travailler directement à partir du backlog du produit. Ca craint, bien sûr, et vous devriez considérer sérieusement de prévoir une nouvelle réunion de planning demain, mais si vous devez vraiment démarrer le sprint alors cela ira probablement. Pour être honnête, toutefois, je n'ai jamais réellement démarré un sprint avec aussi peu d'informations.

Priorité 2 : Liste des histoires que l'équipe a acceptées pour ce sprint.

Priorité 3 : Estimations remplies pour chaque histoire du sprint.

Priorité 4 : « Comment démontrer » rempli pour chaque histoire du sprint.

Priorité 5 : Les calculs de vélocité et de ressources, pour vérifier la vraisemblance de votre plan de sprint. Cela inclue la liste des membres de l'équipe et leurs engagements (sinon vous ne pouvez pas calculer la vélocité).

Gardez les simples et à haut niveau, pour prendre au plus cinq minutes. Demandez : « De la perspective des ressources, y a-t-il quelque chose d'énormément différent dans ce sprint par rapport aux sprints passés ? » Si non, utilisez la météo de la veille. Si oui, faites les ajustements en conséquence.

Priorité 6 : une heure et un lieu bien définis pour la mêlée quotidienne. Cela prend juste un moment pour décider, mais si vous manquez de temps le Scrum master peut simplement le décider après la réunion et envoyer un mail à tout le monde.

Priorité 7 : Les histoires découpées en tâches. Ce découpage peut sinon être fait quotidiennement en conjonction avec les mêlées quotidiennes, mais cela va légèrement perturber le déroulement du sprint.

Les histoires techniques

Voici un point complexe : les histoires techniques. Ou encore éléments non-fonctionnels ou quoi que ce soit que vous vouliez les appeler.

Je ne peux pas m'empêcher de glousser lorsque quelqu'un parle d'exigences non-fonctionnelles". Cela ressemble tellement à « des choses qui ne devraient pas fonctionner ». :o)

Je fais référence aux choses qui doivent être faites mais qui ne sont pas livrables, pas liées directement à des histoires spécifiques, et qui n'ont pas de valeur directe pour le directeur de produit.

Nous les appelons « histoires techniques ».

Par exemple :

- **Installer un serveur de construction (*build*) en continu**
  - Pourquoi il faut le faire : parce que cela économise d'énormes quantités de temps pour les développeurs et réduit le risque de problèmes d'intégration de type big-bang à la fin d'une itération.
- **Rédiger une vue d'ensemble de la conception du système**
  - Pourquoi il faut le faire : parce que les développeurs oublient sans arrêt la conception globale, et par conséquent écrivent du code incohérent. Il faut un document qui montre une vue d'ensemble pour garder tout le monde synchronisé sur la conception.
- **Remanier (*refactor*) la couche d'accès aux données (DAO)**
  - Pourquoi il faut le faire : parce que la couche d'accès est devenue vraiment embrouillée et coûte du temps à tout le monde à cause de la confusion et des bugs pas nécessaires. Nettoyer le code va faire économiser du temps à tout le monde et va améliorer la robustesse du système.
- **Mettre à jour Jira** (système de suivi de bugs)
  - Pourquoi il faut le faire : la version actuelle est trop buggée et lente, mettre à jour fera économiser du temps à tout le monde.

S'agit-il d'histoires au sens habituel ? Ou bien s'agit-il de tâches qui ne sont pas connectées à une histoire particulière ? Qui les priorise ? Est-ce que le directeur de produit devrait être impliqué dans ces choses ?

Nous avons beaucoup expérimenté avec différentes manières de gérer les histoires techniques. Nous avons essayé de les traiter comme des histoires de première classe, tout comme n'importe quelle autre. Cela n'allait pas, en effet quand le directeur de produit priorisait le backlog de produit c'était comme comparer des pommes avec des oranges. En fait, pour des raisons évidentes, les histoires techniques recevaient souvent une faible priorité avec des motivations du type « oui les gars, je suis sûr qu'un serveur de construction en continu est important et tout, mais commençons d'abord par construire des fonctionnalités qui rapportent des sous, n'est-ce pas ? Ensuite vous pourrez ajouter votre petit plaisir technique, OK ? ».

Dans certains cas le directeur de produit a raison, mais souvent il a tort. Nous avons conclus que le directeur de produit n'est pas toujours qualifié pour établir le bon compromis. Donc voici ce que nous faisons :

1) Essayer d'éviter les histoires techniques. Rechercher vraiment un moyen de transformer une histoire technique en histoire normale avec une valeur métier mesurable. De cette manière le directeur de produit a de meilleures chances de faire des compromis corrects.
2) S'il n'est pas possible de transformer une histoire technique en histoire normale, voir si le travail peut être fait en tant que tâche dans une autre histoire. Par exemple « remanier la couche d'accès aux données » pourrait être une tâche dans l'histoire « éditer l'utilisateur », puisqu'elle implique la couche d'accès.
3) Si les deux méthodes ci-dessus échouent, définir une histoire technique, et maintenir une liste séparée de telles histoires. Permettre au directeur de produit de la voir mais pas de l'éditer. Utiliser les paramètres « facteur de focalisation » et « vélocité estimée » pour négocier avec le directeur de produit et obtenir un peu de temps dans le sprint pour implémenter les histoires techniques.

Je trouve toujours les histoires techniques comme étant un bon modèle et je les utilise beaucoup. Les plus petites histoires techniques sont juste embarquées dans le travail au jour le jour, alors que les plus grandes sont écrites et placées dans un backlog d'histoires techniques, visible du product owner mais géré par l'équipe. L'équipe et le product owner se mettent d'accord sur une directive telle que « 10-20% de notre temps est passé sur les histoires techniques ». Pas besoin de plan élaboré de suivi comme le facteur de focalisation ou de fiches de temps, utilisez juste votre intuition. Demandez à la rétro, « Grossièrement, quelle capacité de notre sprint avons-nous dépensé sur des histoires techniques, et est-ce que cela semblait correct ? »

Exemple (un dialogue très similaire à celui-ci s'est passé durant l'un de nos plannings de sprint).

- **Equipe :** « Nous avons des trucs techniques à faire. Nous voudrions passer 10% de notre temps à ça, c'est-à-dire réduire le facteur de focalisation de 75% à 65%. Est-ce OK ? »

- **Directeur de produit :** « Surement pas ! Nous n'avons pas le temps ! »

- **Equipe :** « Eh bien, regardez le sprint dernier (toutes les têtes se tournent vers les gribouillages de vélocité sur le tableau blanc). Notre

vélocité estimée était de 80, et notre vélocité réelle était de 30, n'est-ce pas ? »

- **Directeur de produit :** « Exactement ! Donc nous n'avons pas de temps pour faire des trucs techniques internes ! On a besoin de nouvelles fonctionnalités ! »

- **Equipe :** « Eh bien, la *raison* pour laquelle notre vélocité s'est avérée si mauvaise était que nous avons passé énormément de temps à essayer d'assembler des versions cohérentes pour le test ».

- **Directeur de produit :** « Oui, et alors ? »

- **Equipe :** « Eh bien, notre vélocité va probablement continuer à être aussi mauvaise si nous ne faisons pas quelque chose. »

- **Directeur de produit :** « Oui, et alors ? »

- **Equipe :** « Donc nous proposons de prendre environ 10% de ce sprint pour mettre en place un serveur de construction en continu et d'autres choses qui vont enlever cette épine de l'intégration. Cela va probablement augmenter notre vélocité d'au moins 20% pour chaque sprint suivant, indéfiniment ! »

- **Directeur de produit :** « Vraiment ? Alors pourquoi ne l'avons-nous pas fait le sprint dernier ? »

- **Equipe :** « Euh… parce que vous n'avez pas voulu… »

- **Directeur de produit :** « Oh, hum, très bien, alors on dirait que c'est une bonne idée de le faire maintenant ! »

Bien sûr l'autre option est de laisser le directeur de produit en dehors du circuit ou de lui donner un facteur de focalisation non négociable. Mais il n'y a aucune excuse pour ne pas *essayer* de d'abord atteindre un consensus.

Si le directeur de produit est un gars compétent et raisonnable (et nous avons eu de la chance ici) je suggère de le garder aussi informé que possible, et de le laisser décider les priorités globales. La transparence est l'une des valeurs centrales de Scrum, n'est-ce-pas ?

Si vous ne pouvez pas avoir de franches conversations avec le product owner au sujet de choses comme les histoires techniques, la qualité, et la dette technique, alors vous avez un problème plus profond qui a vraiment besoin d'être adressé !

Un symptôme de cela est si vous vous retrouvez à délibérément cacher des informations au product owner. Vous n'avez pas à impliquer le product owner dans chaque conversation, mais la relation devrait vraiment être basée sur la confiance et le respect ; sans cela, vous avez peu de chances de réussir quel que soit ce que vous construisez.

Système de suivi de bug vs. backlog de produit

Voici un point épineux. Excel est un excellent support pour le backlog de produit. Mais vous avez tout de même besoin d'un système de suivi de bugs, et Excel ne fera probablement pas l'affaire. Nous utilisons Jira.

Donc comment amenons-nous des entrées Jira dans la réunion de planning de sprint ? Je veux dire que ca ne le ferait pas de simplement les ignorer et de se focaliser uniquement sur les histoires.

Nous avons essayé plusieurs stratégies :

1) Le directeur de produit imprime les entrées Jira de plus haute priorité, les apporte à la réunion de planning de sprint, et les met sur le mur avec les autres histoires (et donc spécifie implicitement la priorité de ces éléments par rapport aux autres histoires).
2) Le directeur de produit crée des histoires qui font référence aux entrées Jira. Par exemple « Corriger les bugs les plus critiques sur les rapports back office, Jira-124, Jira-126 et Jira-180 ».
3) La correction de bugs est considérée comme étant en dehors du sprint, c'est-à-dire que l'équipe garde un facteur de focalisation suffisamment faible (par exemple 50%) pour garantir qu'ils ont du temps pour corriger les bugs. On suppose alors que l'équipe va passer un certain temps chaque sprint à corriger des bugs enregistrés dans Jira.
4) Mettre le backlog de produit dans Jira (c'est-à-dire enterrer Excel). Traiter les bugs comme n'importe quelle autre histoire.

Nous n'avons pas réellement conclu quelle stratégie est la meilleure pour nous ; en fait cela varie d'équipe en équipe et de sprint en sprint. J'ai toutefois tendance à préférer la première stratégie. Elle est simple et élégante.

Huit ans plus tard, je ne peux qu'être d'accord. Il n'y a pas une seule meilleure pratique ; chaque stratégie ci-dessus peut être bonne selon le contexte. Expérimentez jusqu'à ce que vous trouviez ce qui marche le mieux pour vous.

La réunion de planning de sprint est finalement terminée !

Waouh, je n'aurais jamais pensé que ce chapitre sur les réunions de planning de sprint serait aussi long ! Je devine que cela reflète mon opinion que la réunion de planning de sprint est la chose la plus importante que vous faites dans Scrum. Faites beaucoup d'efforts pour faire ca correctement, et le reste sera tellement plus facile.

Ou vice versa – faites le reste correctement et le planning de sprint sera du gâteau. :o)

La réunion de planning de sprint est réussie si tout le monde (tous les membres de l'équipe et le directeur de produit) sortent de la réunion avec le sourire, et se lèvent le matin suivant avec le sourire, et font leur première mêlée quotidienne avec le sourire.

Ensuite, bien sûr, toutes sortes de choses peuvent se passer horriblement mal, mais au moins vous ne pourrez pas blâmer le plan du sprint :o)

5

Comment nous communiquons sur les sprints

Il est important de tenir au courant toute l'entreprise sur ce qui se passe. Autrement les gens se plaindront ou, pire, feront de fausses hypothèses sur ce qui se passe.

Pour cela nous utilisons une « page d'information de sprint ».

Equipe Jackass, sprint 15

Objectif du sprint
- Release prête pour la beta !

Backlog du sprint (estimations entre parenthèses)
- Dépôt (3)
- Outil de migration (8)
- Login backoffice (5)
- Administrateur backoffice (5)

Vélocité estimée : 21

Planning
- Période de sprint : 06/11/2006 au 24/11/2006
- Scrum quotidien : 9h30 - 9h45, salle de l'équipe
- Démo du sprint : 24/11/2006 à 13h, cafétéria

Equipe
- Jim
- Erica (scrum master)
- Tom (75%)
- Eva
- John

Parfois nous ajoutons aussi des informations qui expliquent comment chaque user story sera démontrée.

Juste après la réunion de planification d'itération, le ScrumMaster crée cette page, la dépose sur le wiki, et envoie un spam à toute l'entreprise.

Objet : sprint 15 de Jackass a commencé

Salut à tous ! L'équipe Jackass a commencé le sprint 15. Notre but est de démontrer une release prête pour la beta le 24 novembre.

Les détails sur la page d'info du sprint : http://wiki.mycompany.com/jackass/sprint15

Nous avons aussi un « tableau de bord » sur notre wiki qui donne des liens vers toutes les itérations en cours.

Tableau de bord

Sprints en cours

- Equipe X sprint 15
- Equipe Y sprint 12
- Equipe Z sprint 1

En plus, le ScrumMaster imprime la page d'information de sprint et l'affiche sur le mur extérieur du bureau de l'équipe. De cette façon n'importe qui passant par là peut y jeter un œil pour savoir ce que cette équipe est en train de faire. Comme cela indique aussi où et quand ont lieu la mêlée quotidienne et la démonstration de sprint, cette personne sait où aller pour obtenir plus d'information.

Quand l'itération approche de la fin, le ScrumMaster rappelle à tout le monde la prochaine démo.

Objet : démo sprint Jackass demain 13h à la cafétéria

Salut à tous ! Vous êtes les bienvenus pour assister à la démo de sprint à 13h à la cafétéria demain (vendredi). Nous démontrerons une release prête pour la beta.

Les détails sur la page d'info du sprint : http://wiki.mycompany.com/jackass/sprint15

Avec tout cela, personne n'a vraiment d'excuse pour ne *pas* savoir ce qui se passe.

Hé, quelle bonne idée ! Je devrais faire ça plus souvent !

6

Comment nous faisons les backlogs de sprint

Vous êtes arrivé jusque là ? Ouf, bon travail.

Maintenant que nous avons terminé la réunion de planification du sprint et parlé au monde entier de notre superbe nouveau sprint, il est temps pour le Scrum master de créer un backlog de sprint. Cela doit être fait *après* la réunion de planification du sprint, mais *avant* la première mêlée quotidienne.

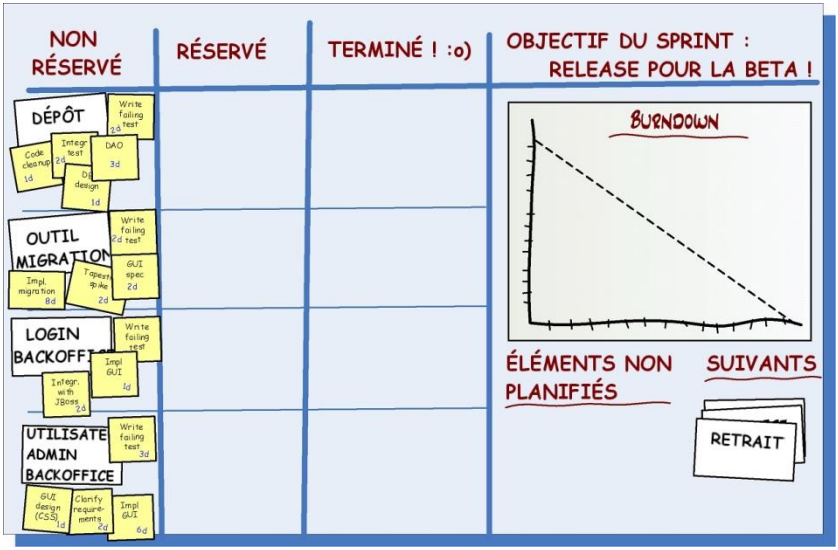
Format du backlog de sprint

Nous avons essayé différents formats pour le backlog de sprint, y compris Jira, Excel et un tableau physique des tâches sur le mur. Au début, nous avons utilisé Excel le plus souvent, il y a à disposition du public de nombreux modèles Excel pour les backlogs de Sprint, avec génération automatique de burndown chart et des choses comme ça. Je pourrais discuter longtemps sur la façon dont nous avons amélioré notre backlog de Sprint sous Excel. Mais je ne le ferai pas. Je ne vais même pas mettre un exemple ici.

A la place je vais vous décrire en détail ce que nous considérons être le format le plus efficace pour les backlogs de sprint - un tableau des tâches sur le mur !

Ouep, les tableaux de tâches sur les murs (alias tableaux Scrum) sont toujours mon outil de premier choix ! Etonnamment puissant. Pour une équipe distribuée, utilisez un outil qui fournit une vue de tableau Scrum (ce qu'à peu près tous les outils font), et mettez le sur grand écran dans chaque site. A la mêlée quotidienne, tout le monde se tient devant l'écran du mur et parle via Skype (ou équivalent).

Trouvez un grand mur inutilisé ou contenant des choses inutiles comme le logo de la société, des vieux diagrammes ou d'affreuses peintures. Nettoyez le mur (demandez la permission si vous le devez). Scotchez une grande, grande feuille de papier (au moins 2 x 2 mètres, ou 3 x 2 mètres pour une grande équipe). Puis faites ceci :



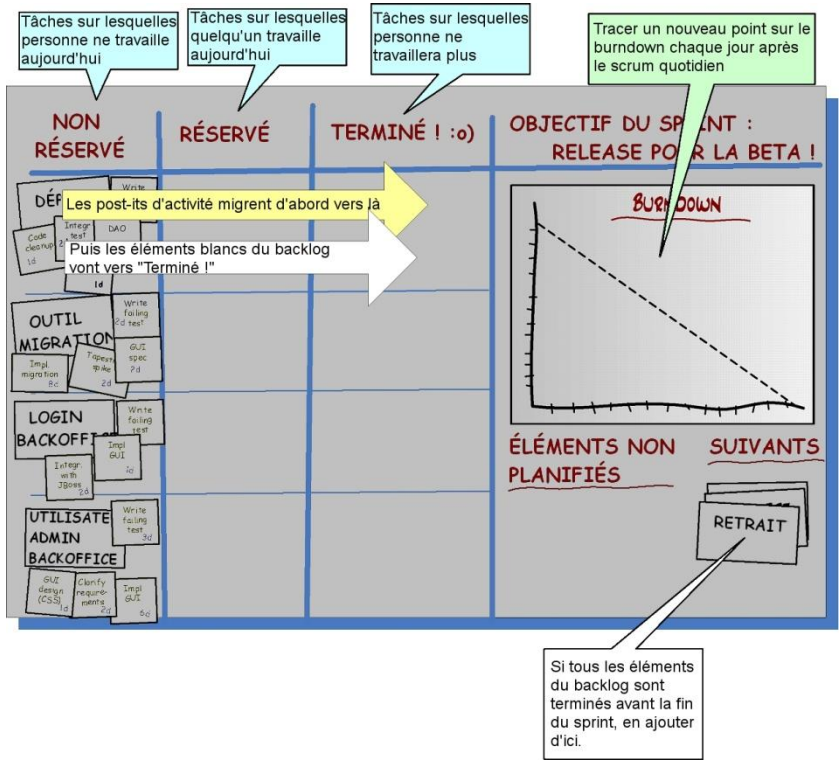
Vous pourriez utiliser un tableau blanc bien sûr. Mais ce serait un peu du gâchis. Si possible, gardez les whiteboards pour les brouillons de conception et utilisez les murs sans whiteboards pour les tableaux des tâches.

Ou encore mieux, remplissez vos murs de tableaux blancs. Un investissement qui en vaut la peine !

NOTE - si vous utilisez des post-its pour les tâches, n'oubliez pas de les attacher avec du vrai scotch, ou vous retrouverez un jour tous vos post-its en tas sur le sol.

Ou achetez juste des post-its super-collants, légèrement plus chers mais ils ne tombent pas ! (Non, je ne suis pas sponsorisé, vraiment.)

Comment le tableau des tâches fonctionne

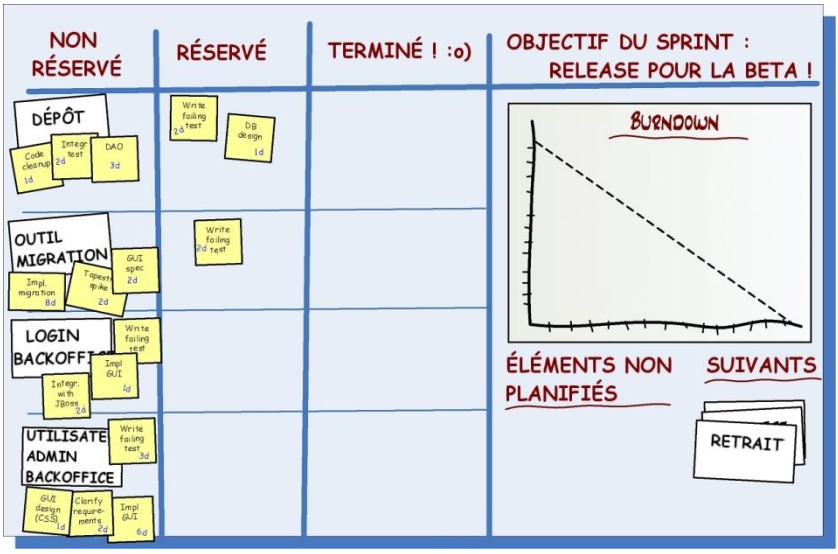


Vous pourriez bien sûr ajouter toute sorte de colonnes. « En attente pour les tests d'intégration » par exemple. Ou « Annulé ». Cependant avant que vous ne compliquiez les choses, réfléchissez longuement. Est-ce que cet ajout est vraiment, *vraiment* nécessaire ?

J'ai trouvé que la simplicité est extrêmement importante pour ce genre de choses, aussi je rajoute de la complexité que lorsque *ne pas* le faire devient trop coûteux.

Exemple 1 - Après la première mêlée quotidienne

Après la première mêlée quotidienne, le tableau des tâches devrait ressembler à ça :



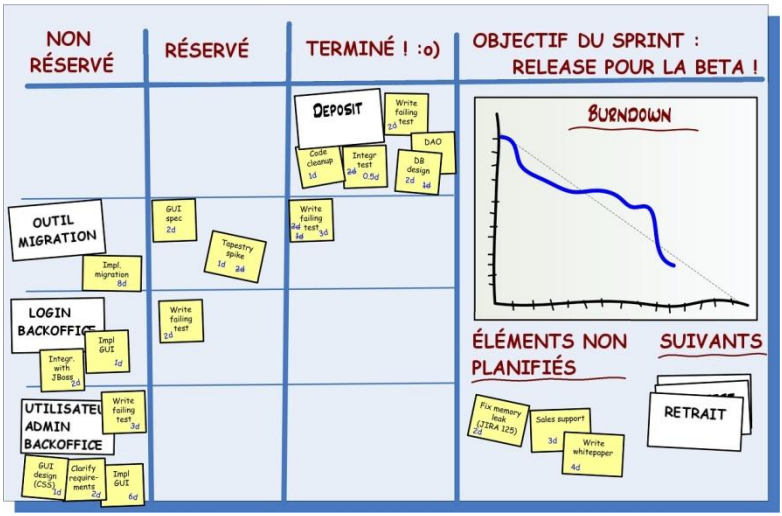
Comme vous pouvez le voir, trois tâches ont été « réservées », c'est-à-dire que l'équipe va travailler dessus aujourd'hui.

Parfois, pour de grandes équipes, une tâche reste coincée à «réservé» car personne ne se souvient qui a travaillé dessus. Si cela arrive fréquemment dans une équipe elle adopte généralement une politique d'étiquetage des tâches réservées en indiquant le nom de la personne qui l'a réservée.

Presque toutes les équipes utilisent des avatars de nos jours. Chaque membre d'équipe choisit son avatar (un personnage de South Park ou autre), l'imprime, et le met sur aimant. Une excellente façon de voir qui travaille sur quoi. Aussi, si chaque personne n'a seulement que deux aimants, cela limite indirectement le travail en cours et le multitâches. « C'est quoi ce bordel, je n'ai plus d'avatars ! » Ouais, alors arrête de commencer et commence à finir tes tâches !

Exemple 2 – après quelques jours

Après quelques jours le tableau des tâches devrait ressembler à ça :



Comme vous pouvez le voir, nous avons terminé l'histoire «déposer» (c'est-à-dire qu'elle a été enregistrée dans le référentiel des sources, testée, refactorée, etc). L'outil de migration est en partie fini, l'histoire «back office login» est commencée, et le «back office user admin» n'est pas entamé.

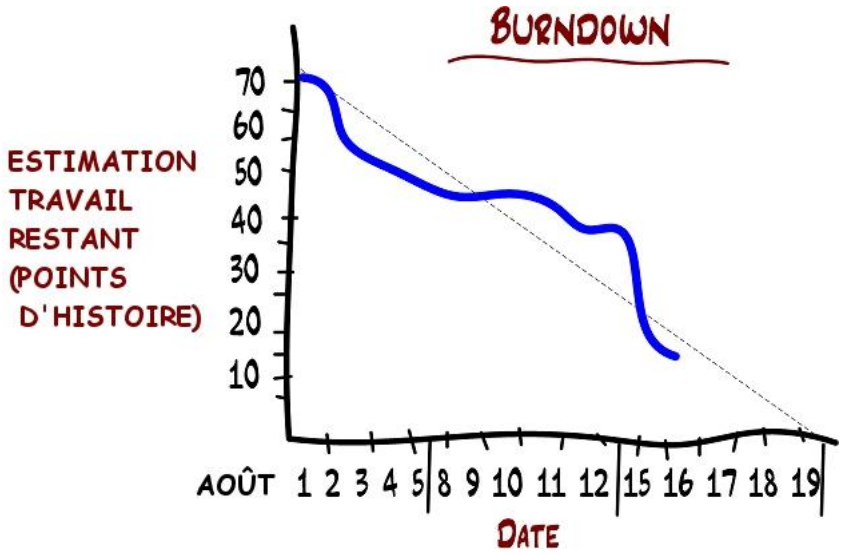
Nous avons 3 éléments non planifiés, comme vous pouvez le voir en bas à droite. C'est utile pour se rappeler ce que l'on a fait pour la rétrospective de sprint.

Ici c'est l'exemple d'un vrai backlog de sprint approchant la fin du sprint. Il devient moins ordonné au fur et à mesure que le sprint progresse, mais c'est bon

tant que ça ne dure pas longtemps. A chaque nouveau sprint nous créons un nouveau backlog de sprint, tout propre.

Comment le burndown chart fonctionne

Regardons de plus près le burndown chart :



Cette courbe montre que :

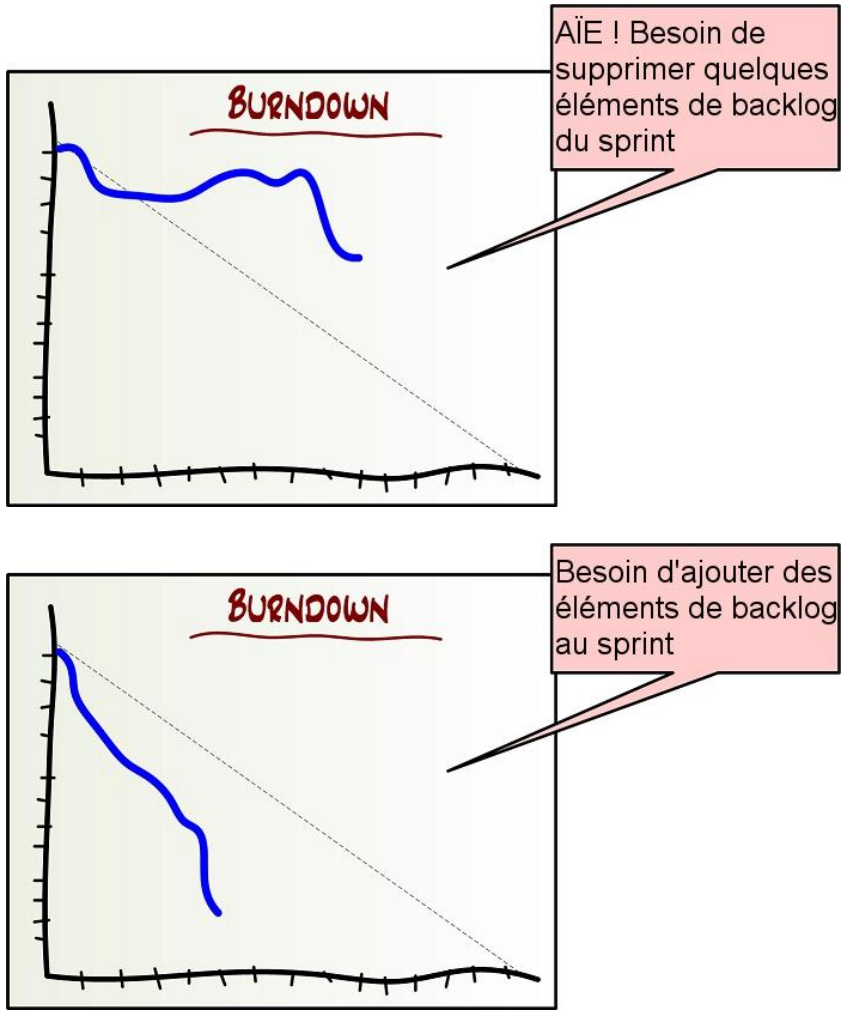
- Le premier jour du sprint, le 1er août, l'équipe a estimé qu'il y avait approximativement 70 points d'histoire de reste à faire. C'est en fait la *vélocité estimée* du sprint entier.
- Le 16 août l'équipe estime qu'il y a approximativement 15 points d'histoire de reste à faire. La ligne de tendance en pointillés montre qu'ils sont sur la bonne voie, c'est-à-dire qu'à ce rythme ils auront tout fini d'ici la fin du sprint.

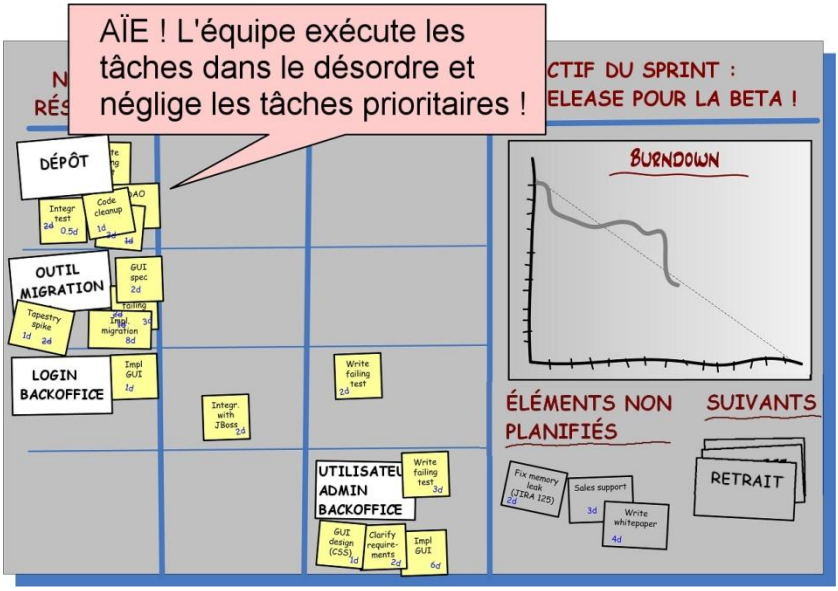
Nous excluons les weekends sur l'axe des abscisses puisque le travail est rarement fait le week-end. Nous avions l'habitude d'inclure les weekends mais cela rend la courbe un peu confuse, car elle «stagne» les weekends ce qui ressemble à un signal d'avertissement.

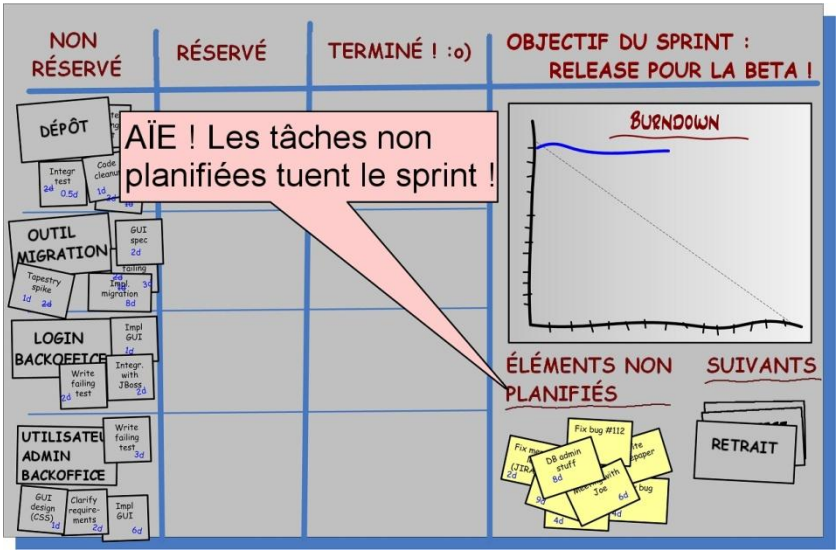
Scrum a été conçu à l'origine pour des équipes faisant des sprints d'un mois et utilisant Excel pour suivre les tâches. Dans ce cas, un burn-up chart est un résumé visuel vraiment utile sur comment on se débrouille. Maintenant, toutefois, la plupart des équipes font des sprints plus courts et ont des tableaux Scrum très visuels. Alors ils mettent de côté de plus en plus les burn-down charts complètement, car un coup d'œil sur le tableau Scrum leur donne l'information dont ils ont besoin. Essayez d'enlever le burn-down chart et voyez s'il vous manque !

Les signaux d'avertissement du tableau des tâches

Un rapide coup d'œil sur le tableau des tâches devrait donner à tout le monde une indication sur le bon déroulement du sprint. Le Scrum master a la responsabilité de s'assurer que l'équipe réagit aux signaux d'avertissement comme :




AÏE ! L'équipe exécute les tâches dans le désordre et néglige les tâches prioritaires !
CTIF DU SPRINT :
ELEASE POUR LA BETA !
DÉPÔT
Integr test 2d
Code cleanup 1d
0.5d
OUTIL MIGRATION
GUI spec 2d
Tapestry spike 1d
2d
Impl. migration 8d
3d
LOGIN BACKOFFICE
Impl GUI 1d
Integr. with JBoss 2d
Write failing test 2d
UTILISATEU ADMIN BACKOFFICE
Write failing test 3d
GUI design (CSS) 1d
Clarify require-ments 2d
Impl GUI 6d
BURNDOWN
ÉLÉMENTS NON PLANIFIÉS
SUIVANTS
Fix memory leak (JIRA 125) 2d
Sales support 3d
Write whitepaper 4d
RETRAIT


NON RÉSERVÉ
RÉSERVÉ
TERMINÉ ! :o)
OBJECTIF DU SPRINT :
RELEASE POUR LA BETA !
AÏE ! Les tâches non planifiées tuent le sprint !
DÉPÔT
OUTIL MIGRATION
LOGIN BACKOFFICE
UTILISATEU ADMIN BACKOFFICE
BURNDOWN
ÉLÉMENTS NON PLANIFIÉS
SUIVANTS
Fix bug #112
DB admin stuff 8d
RETRAIT

Hé, et la traçabilité ?!

La meilleure traçabilité que je peux offrir dans ce modèle est de prendre une photo numérique du tableau des tâches chaque jour. Si vous le devez. Je l'ai fait parfois mais je n'ai jamais eu besoin d'utiliser ces photos.

Si la traçabilité est très importante pour vous, alors peut-être que le tableau des tâches n'est pas la solution qui vous convient.

Mais je suggère d'essayer d'estimer la valeur apportée par une traçabilité détaillée du sprint. Une fois que le sprint est fini et que le code qui marche a été livré et que la documentation a été enregistrée, est-ce que ça intéresse vraiment quelqu'un de savoir combien d'histoires étaient terminées au 5e jour du sprint ? Est-ce que ça intéresse vraiment quelqu'un de savoir quelle était l'estimation de temps pour « écrire un test qui échoue pour Déposer » ?

Je trouve toujours la traçabilité énormément surévaluée. Certains outils fournissent ce type de donnée mais les personnes ne l'utilisent en fait jamais. Votre système de contrôle de version de code vous donnera la plupart des choses dont vous avez besoin (« Bordel ? Qui a fait ce changement ?! »). Ajoutez des conventions simples telles que l'écriture d'ID d'histoire dans le commentaire de commit, et vous obtiendrez bien assez de traçabilité pour la plupart des contextes.

Estimations en jours vs. heures

Dans la plupart des livres et articles sur Scrum vous trouverez que les tâches sont estimées en heures, pas en jours. Nous avions l'habitude de faire ainsi. Notre formule générale était : 1 jour-homme effectif = 6 heures-homme effectives.

Maintenant nous avons arrêté de faire ça, au moins dans la plupart de nos équipes, pour les raisons suivantes :

- Les estimations en heures-homme sont trop fines, cela engendre trop de petites tâches de 1-2 heures et entraîne par conséquent le micro-management.
- Ca masque le fait que tout le monde pense en jours-homme de toutes façons, et qu'on multiplie juste par 6 avant d'écrire en heures-homme. « Hmmmm, cette tâche devrait prendre à peu près un jour. Oh je dois écrire en heures, j'écrirai 6 heures alors ».
- Deux unités différentes sèment la confusion. « Cette estimation était-elle en heures-homme ou en jours-homme ? ».

Donc maintenant nous utilisons les jours-homme comme base pour toutes nos estimations de temps (bien que nous appelons ça des points d'histoire). Notre plus petite valeur est 0.5, c'est-à-dire que toute tâche inférieure) 0.5 est soit supprimée, soit combine avec une autre tâche, ou simplement laissée avec l'estimation 0.5 (il n'y'a pas grand mal à surestimer un peu). Elégant et simple.

Même plus simple : sautez l'estimation de tâches entièrement ! La plupart des équipes apprennent un jour comment diviser leur travail en tâches de grossièrement un jour pour une ou deux personnes. Si vous pouvez faire ça, vous n'avez pas besoin de vous embêter avec les estimations de tâches, ce qui élimine beaucoup de gaspillage. Les burn-down charts peuvent toujours être utilisés (si vous êtes obligés) – dans ce cas, comptez juste les tâches au lieu de comptabiliser les heures.

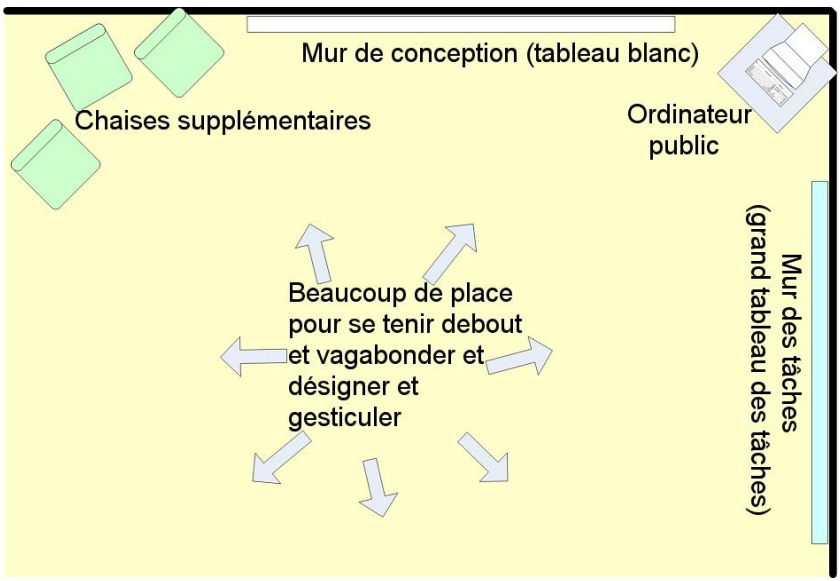
7

Comment nous organisons le bureau de l'équipe

Le coin conception

J'ai noté que beaucoup de discussions de conception les plus intéressantes et ayant le plus de valeur prennent place spontanément en face du tableau des tâches.

Pour cette raison, nous essayons d'arranger cet espace en «coin conception».



C'est vraiment utile. Il n'y a pas de meilleur moyen pour avoir une vue d'ensemble du système qu'en se tenant dans le coin conception, on peut jeter un œil sur les deux murs, puis jeter un œil sur l'ordinateur et essayer le dernier build

du système (si vous avez assez de chance pour avoir une intégration continue, allez voir p.100 « Comment nous combinons Scrum et XP »).

Le « mur de conception » est juste un grand tableau blanc contenant les schémas et les impressions des documents de conception les plus importants (diagrammes de séquences, prototypes IHM, modèles métier, etc).

Ci-dessus : une mêlée quotidienne se passant dans le coin mentionné plus haut.

Hmmmm..... ce burndown semble trop parfait et trop droit, vous ne trouvez pas ? Mais l'équipe insiste qu'il est authentique :o)

Le Coach Malveillant fournit de fausses règles de burn-down. Très commode pour les environnements très dysfonctionnels. :o)
http://blog.crisp.se/2013/02/15/evil-coach/fake-burndown-ruler

Quand on en vient à l'organisation des bureaux il y a une chose qui n'est jamais assez soulignée.

Installez l'équipe ensemble !

Pour clarifier un peu, je disais

Installez l'équipe ensemble !

Après huit années à aider des sociétés avec Scrum, j'aimerais ajouter :

INSTALLEZ L'EQUIPE ENSEMBLE !

Les gens sont réticents à bouger. Au moins dans les endroits où j'ai travaillé. Ils n'ont pas envie de prendre toutes leurs affaires, débrancher le pc, déplacer toutes leurs poubelles jusqu'à un nouveau bureau, et tout rebrancher. Plus courte est la distance, plus grande est la réticence. «Allez chef, ça sert à quoi de bouger juste de 5 mètres ?»

Quand on construit des équipes Scrum efficaces, cependant, il n'y a pas d'alternative. Mettez juste l'équipe ensemble. Même si vous devez personnellement menacer chaque personne, transporter vous-même tout leur barda, et effacer leurs anciennes tâches de café. S'il n'y a pas de place pour l'équipe, faites-en. N'importe où. Même si vous devez placer l'équipe au sous-sol. Déplacez les tables, soudoyez le responsable des bureaux, faites ce qu'il faut. Mettez juste l'équipe ensemble.

Une fois l'équipe ensemble, les gains sont immédiats. Après le premier sprint déjà l'équipe sera d'accord qu'il s'agissait d'une bonne idée de regrouper l'équipe (d'expérience personnelle, il n'y a rien qui dit que votre équipe ne sera pas trop bornée pour l'admettre).

Maintenant qu'est-ce que «ensemble» signifie ? Comment doit-on disposer les bureaux ? Bien, je n'ai pas d'opinion tranchée sur la disposition optimale des bureaux. Et même si j'en avais, je suppose que la plupart des équipes n'ont pas le luxe d'être capable de décider exactement comment disposer leurs bureaux. Il y a

habituellement des contraintes physiques – l'équipe voisine, la porte des toilettes, la grosse machine au milieu du bureau, peu importe.

«Ensemble» signifie :

- **Audibilité :** N'importe qui dans l'équipe peut discuter avec quelqu'un d'autre sans crier ou sortir du bureau.
- **Visibilité :** Tout le monde peut voir tout le monde. Tout le monde peut voir le tableau des tâches. Pas forcément assez près pour pouvoir le *lire*, mais au moins le *voir*.
- **Isolation :** Si toute votre équipe se lève soudainement et engage une discussion spontanée et animée sur la conception, il n'y a personne d'extérieur à l'équipe à proximité qui pourrait être perturbé. Et vice versa.

L'»Isolation» ne signifie pas que l'équipe doit être complètement isolée. Dans un environnement de cabines ce serait suffisant si votre équipe avait sa propre cabine et des murs de cabine suffisamment épais pour filtrer la *plupart* du bruit extérieur.

Et si vous avez une équipe distribuée ? Vous n'avez pas de chance. Utilisez autant de moyens techniques que vous pouvez pour minimiser les dégâts – vidéo conférence, webcams, outils de partage de bureau, etc.

Garder le directeur de produit à distance

Le directeur de produit doit être suffisamment près pour que l'équipe puisse aller le voir et lui demander quelque chose, et pour qu'il puisse se promener autour du tableau des tâches. Mais il ne doit pas s'asseoir avec l'équipe. Pourquoi ? Parce qu'il y a des chances qu'il ne soit pas capable de s'empêcher de rentrer dans les détails, et l'équipe ne «prendra pas forme» comme il faut (c'est-à-dire atteindre une équipe soudée, autogérée, hyper productive).

Pour être honnête, c'est de la spéculation. Je n'ai pas encore vu jusque là un cas où le directeur de produit s'assoit avec l'équipe, aussi je n'ai aucune raison empirique pour dire que c'est une mauvaise idée. Juste l'instinct et des ouï-dire d'autres Scrum masters.

Et bien, vous savez quoi. J'avais tort ! Complètement tort. Les meilleures équipes que j'ai vues ont le product owner embarqué. Les équipes souffrent beaucoup quand le product owner est trop loin, et c'est un bien plus gros problème que d'être trop proche. Cependant, le product owner a besoin d'équilibrer son temps entre l'équipe et les parties prenantes, de manière à ce qu'il ne passe pas TOUT son temps avec l'équipe. Mais, de manière générale, le plus proche sera le mieux.

Garder les directeurs et les coachs à distance

C'est un peu dur pour moi d'écrire à ce propos, puisque j'ai été directeur et coach...

C'était mon job de travailler aussi près que possible avec l'équipe. Je montais les équipes, je naviguais entre elles, je faisais du programmation en binôme avec certaines personnes, je coachais des Scrum masters, j'organisais les réunions de planification de sprint, etc. Rétrospectivement la plupart des gens pensent que c'était une bonne chose, car j'avais de l'expérience avec le développement logiciel agile.

Mais, ensuite, j'étais (démarrez la musique de Dark Vador) le responsable des développements, un rôle de responsable fonctionnel. Ce qui signifie pour une équipe qu'elle devient automatiquement moins auto-organisée. «Heck, le chef est là, il a probablement des tas d'avis sur ce qu'on devrait faire et qui devrait le faire. Je vais le laisser parler.»

Mon avis est ; si vous êtes Scrum coach (et peut-être aussi un directeur), impliquez vous autant que possible. Mais seulement pour une période limitée, puis effacez-vous et laissez l'équipe prendre forme et s'autogérer. Surveillez l'équipe de temps en temps (pas trop souvent) en assistant aux démos de sprint et

en regardant le tableau des tâches et en écoutant aux mêlées quotidiennes. Si vous voyez des points à améliorer, prenez le Scrum master à part et conseillez-le. *Pas* devant l'équipe. Une autre bonne idée est d'assister aux rétrospectives de sprint (voir p.82 «Comment nous faisons les rétrospectives de sprint»), Si votre équipe vous fait confiance, ne laissez pas votre présence les intimider.

Pour que les équipes Scrum fonctionnent bien, assurez-vous qu'ils ont tout ce qu'il faut, puis restez à l'écart (à part pendant les démos de sprint).

Cette dernière phrase est toujours le meilleur conseil d'ensemble que j'ai pour les managers dans un contexte agile. Les bons managers sont un facteur de succès crucial, mais en tant que manager, vous devez essayer de vous rendre redondant. Vous ne réussirez probablement pas, mais le seul acte d'essayer va vous pousser dans la bonne direction. Demandez-vous « De quoi cette équipe a-t'elle besoin pour se gérer elle-même ? » plutôt que « Comment puis-je gérer cette équipe ? » Ils ont besoin de choses comme la transparence, un objectif clair, un environnement fun et motivant, de support, et un processus d'escalade pour les impédimentas.

8

Comment nous faisons les mêlées quotidiennes

Nos mêlées quotidiennes suivent à peu près le livre. Elles commencent exactement à l'heure, chaque jour à la même place. Au début nous allions dans une pièce séparée pour faire le sprint planning (à l'époque où nous utilisions des backlogs de sprint électroniques), mais maintenant nous faisons les mêlées quotidiennes dans le bureau de l'équipe pile en face du tableau des tâches. Il n'y a rien de mieux.

Nous faisons les mêlées en nous tenant debout, ce qui diminue le risque de dépasser les 15 minutes.

La mêlée quotidienne est vraiment importante ! C'est le moment où la plupart de la synchronisation se passe et où l'équipe remonte les impédimentas importants. Néanmoins, mal faite, elle peut être vraaaiiiment ennuyeuse – un groupe de gens qui déblatèrent et personne qui n'écoute vraiment.

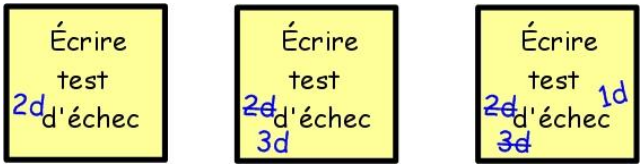
Le Guide Scrum a récemment mis à jour les trois questions pour contrer cela :

- Qu'ai-je réalisé hier qui a aidé notre équipe à atteindre l'objectif de sprint ?
- Que vais-je faire aujourd'hui qui aidera notre équipe à atteindre l'objectif de sprint ?
- Est-ce que je vois des impédimentas qui m'empêchent moi ou notre équipe d'atteindre l'objectif du sprint ?

Remarquez l'attention sur l'objectif de sprint, le but haut niveau partagé de l'équipe ! Si vos mêlées quotidiennes commencent à ternir, essayez ces questions ! Ou peut-être la version de Dan North : demandez « Quelle serait la meilleure journée que nous pourrions avoir ? » suivie d'une discussion ouverte. Peu importe ce que vous faites, ne laissez pas des mêlées quotidiennes rester ennuyeuses. Continuez à expérimenter !

Comment nous mettons à jour le tableau des tâches

Nous mettons à jour le tableau des tâches pendant la mêlée quotidienne. Pendant que chaque personne décrit ce qu'elle a fait hier et ce qu'elle va faire aujourd'hui, elle déplace les post-its sur le tableau des tâches. Quand elle décrit une tâche non planifiée, elle colle un post-it pour ça. Quand la personne met à jour son estimation de temps, elle écrit la nouvelle estimation sur le post-it et raye l'ancienne valeur. Parfois le Scrum Master s'occupe des post-its pendant que les personnes parlent.



Certaines équipes adoptent comme politique de mettre à jour le tableau des tâches *avant* chaque mêlée. Cela fonctionne aussi. Choisissez juste une politique est tenez-vous y.

Beaucoup d'équipes passent un temps démesuré à mettre à jour des chiffres sur des post-its pendant la mêlée quotidienne. Gâchis ! Le but de la mêlée quotidienne est de se synchroniser, alors je trouve généralement mieux de mettre à jour le tableau « en temps réel » (c.à.d le jour où les trucs arrivent) et de sauter l'estimation des tâches complètement. De cette manière la mêlée quotidienne est utilisée pour en effet *communiquer* plutôt qu'administrer.

Indépendamment du format de votre backlog de sprint, essayez d'impliquer *toute l'équipe* en gardant le backlog de sprint à jour. Nous avons essayé des sprints où le Scrum Master est le seul à maintenir le backlog de sprint et où il doit faire le tour chaque jour et demander aux gens leurs estimations de reste à faire. Les inconvénients de cette méthode:

- Le Scrum master passe trop de temps à s'occuper des tâches administratives, au lieu de supporter l'équipe et d'enlever des obstacles.
- Les membres de l'équipe ne sont pas au courant du statut du sprint, puisqu'ils n'ont pas besoin de prêter attention au backlog de sprint. Ce manque de feedback réduit l'agilité globale et la concentration de l'équipe.

Si le backlog de sprint est bien conçu il devrait être aussi facile pour chaque membre de le mettre à jour lui-même.

Immédiatement après la mêlée quotidienne, quelqu'un additionne toutes les estimations de temps (en ignorant celles de la colonne «terminé» bien sûr) et marque un nouveau point sur la courbe du reste à faire du sprint.

Traiter avec les retardataires

Certaines équipes ont un pot pour les pièces et billets. Quand vous êtes en retard, même si c'est juste une minute, vous ajoutez un montant fixe au pot. Aucune question n'est posée. Même si vous appelez avant la réunion pour prévenir que vous serez en retard, vous devrez quand même payer.

Vous échappez à la sentence seulement si vous avez une bonne excuse comme un rendez-vous chez le médecin ou votre propre mariage ou quelque chose dans le genre.

L'argent dans le pot est utilisé pour les évènements collectifs. Pour acheter des hamburgers quand nous passons des nuits à jouer par exemple :o)

Ca marche bien. Mais c'est nécessaire seulement pour les équipes où les gens sont souvent en retard. Certaines équipes n'ont pas besoin de ce type de pratique.

Les équipes utilisent toutes sortes de combines pour se taquiner entre-elles pour que les personnes soient à l'heure (si besoin). Assurez-vous juste que l'équipe vienne à cela par elle-même ; n'imposez pas de stratagèmes d'au-dessus ou hors de l'équipe. Et gardez celà amusant. Dans une équipe, les retardataires devaient chanter une chanson stupide. Si vous étiez en retard une seconde fois, vous deviez accompagner cela avec les mouvements de danse correspondants. :o)

Traiter avec « je ne sais pas quoi faire aujourd'hui »

Ce n'est pas rare pour quelqu'un de dire « Hier j'ai fait bla bla bla, mais aujourd'hui je n'ai pas la moindre idée de ce que je peux réaliser » (hé ce dernier vers rime). Maintenant quoi ?

Disons que Joe et Lisa sont ceux qui ne savent pas quoi faire aujourd'hui.

Si je suis Scrum master je passe et laisse la personne suivante parler, mais je note qui n'a rien à faire. Après que tout le monde ait parlé, je repasse sur le tableau des tâches avec toute l'équipe, de haut en bas, je vérifie que tout est synchro, que tout le monde sait ce que chaque élément signifie, etc. J'invite les gens à rajouter des post-its. Puis je reviens vers ceux qui ne savaient pas quoi faire «maintenant que nous avons passé en revue le tableau des tâches, avez-vous une idée de ce que vous pourriez faire aujourd'hui»? Avec un peu de chance ils ont une idée.

Sinon, je considère s'il n'y a pas une opportunité de programmation en binôme ici. Disons que Niklas est en train d'implémenter l'IHM d'admin du back-office aujourd'hui. Dans ce cas je suggère poliment que peut-être Joe ou Lisa pourrait programmer en binôme avec Niklas sur ce sujet. Ca fonctionne en général.

Et si ça ne fonctionne pas, voici l'astuce suivante.

Scrum master : « OK, qui veut nous montrer la version beta de l'application ? » (en supposant que c'est le but de l'itération)

L'équipe : silence confus

Scrum master : « Nous ne sommes pas prêts ? »

L'équipe: « hmm... non »

Scrum master : « Ah bon. Pourquoi ? Qu'est-ce qu'il reste à faire ? »

L'équipe : « Ben nous n'avons même pas de serveur de test pour la lancer, et le script de build est cassé. »

Scrum master : « Aha. » (ajoutez deux post-its sur le mur des tâches). « Joe et Lisa, comment pouvez-vous nous aider aujourd'hui ? »

Joe : « Hmm.... au hasard je vais essayer de trouver un serveur de test quelque part ».

Lisa : « ... et je vais essayer de réparer ce script de build ».

Si vous avez de la chance, quelqu'un va effectivement faire la démo de la version beta que vous avez demandée. Super ! Vous avez atteint le but de votre sprint. Mais que faire si vous êtes au milieu du sprint ? Facile. Félicitez l'équipe pour le travail accompli, prenez une ou deux histoires de la section «à venir» en bas à droite du tableau des tâches, et déplacez-les vers la colonne « non réservé ». Puis refaites la mêlée quotidienne. Avertissez le directeur de produit que vous avez ajouté des éléments au sprint.

Ou utilisez le temps pour rembourser de la dette technique, ou faites de l'exploration technique. Gardez le product owner dans la boucle toutefois.

Mais que faire si vous n'avez atteint l'objectif du sprint et que Joe et Lista continuent de refuser de se rendre utile. Je considère habituellement l'une des stratégies suivantes (aucune d'elle n'est élégante, mais il s'agit d'un dernier recours) :

- **La honte :** « Bien si tu n'as aucune idée de comment aider l'équipe, je suggère que tu rentres chez toi, ou que tu lises un bouquin ou quelque chose d'autre. Ou assied-toi jusqu'à ce quelqu'un t'appelle pour l'aider. »
- **Vieille école :** Assignez-leur simplement une tâche.
- **Pression collective :** Dîtes « prenez tout votre temps Joe et Lisa, nous resterons là jusque vous trouviez quelque chose à faire qui peut nous aider à atteindre l'objectif »
- **Servitude :** Dîtes « Bien, vous pouvez aider l'équipe indirectement en étant les majordomes aujourd'hui. Faites le café, donnez des massages, nettoyez les poubelles, préparez le déjeuner, et tout ce que nous pourrions demander durant la journée. » Vous pourriez être surpris par la vitesse à laquelle Joe et Lisa réussissent à trouver des tâches techniques utiles :o)

Si une personne vous force fréquemment à aller si loin, vous devriez probablement prendre cette personne à part et faire du coaching poussé. Si le problème persiste, vous devez évaluer si cette personne est importante pour votre équipe ou pas.

Si elle *n'est pas* importante, essayez de la dégager de votre équipe.

Si elle *est* importante, alors essayez de la mettre en binôme avec quelqu'un qui sera son « berger ». Joe est peut-être un bon développeur ou architecte, c'est juste qu'il préfère vraiment que quelqu'un d'autre lui dise quoi faire. Bien. Donnez à Niklas la responsabilité d'être le berger permanent de Joe. Ou prenez cette responsabilité vous-même. Si Joe est assez important pour votre équipe

l'effort sera moindre. Nous avons eu des cas comme ça et ça marchait plus ou moins bien.

Le problème du "je ne sais pas quoi faire aujourd'hui » est typique pour les équipes qui sont novices à Scrum, et qui sont habituées à avoir d'autres personnes décidant les choses pour eux. Au fur et à mesure qu'elles accumulent de l'expérience en auto-organisation, le problème disparait. Les personnes apprennent à déterminer quoi faire. Alors si vous êtes un Scrum master et que vous vous trouvez à recourir aux astuces ci-dessus trop souvent, vous devriez envisager à prendre du recul. Malgré votre intention d'aide, vous pourriez être le plus gros impédimenta de l'équipe, les empêchant d'apprendre comment s'auto-organiser !

9

Comment nous faisons les démos de sprint

La démo de sprint (ou revue de sprint comme certains l'appellent) est une partie importante de Scrum que les gens ont tendance à sous-estimer.

« Oh *devons*-nous vraiment faire cette démo ? Il n'y vraiment pas grand-chose d'amusant à montrer ! »

« Nous n'avons pas le temps de préparer une &%$# de démo! »

« Je n'ai pas le temps d'assister aux démos des autres équipes ! »

Je ne peux pas comprendre pourquoi j'ai appelé cela la "démo de sprint"! A quoi est-ce que je pensais ? Le terme officiel est la « revue de sprint », et un terme bien meilleur. Démo implique une communication à sens unique (« voilà, c'est ce que nous avons construit »), alors que revue implique une communication à double sens (« voilà ce que l'on a construit, qu'est ce que vous en pensez ? »). La revue de sprint est avant tout une question de feedback ! Alors quand vous lirez « démo » plus bas, pensez « revue », OK ?

Pourquoi nous insistons pour que tous les sprints finissent par une démo

Une démo de sprint bien faite, bien que ça puisse sembler peu marquant, a un effet profond.

- L'équipe se voit attribuer le mérite pour le travail accompli. Ils *se sentent bien*.
- Les autres personnes découvrent ce que l'équipe fait.
- La démo donne un retour vital de la part des intervenants.
- Les démos sont (ou devraient) être un évènement social où les différentes équipes peuvent interagir entre elles et discuter de leur travail. Cela a de la valeur.
- Faire une démo force l'équipe à *terminer leur travail* et à le livrer (même si c'est seulement pour environnement de test). Sans démo, nous garderions une énorme pile de choses terminée à 99%. Avec les démos nous avons peut-être moins d'éléments terminés, mais ils sont

réellement terminés, ce qui est (dans notre cas) une bien meilleure chose que d'avoir une pile entière de tâches qui sont à peu près *terminés* et qui pollueront le prochain sprint.

Si l'équipe est plus ou moins obligée de faire une démo, si elle n'a pas quelque chose qui fonctionne vraiment, la démo sera embarrassante. L'équipe bégaiera et hésitera pendant la démo et les applaudissements ne seront qu'à moitié sincères. Les gens se sentiront désolés pour l'équipe, et seront peut-être irrité d'avoir gâché du temps pour assister à une démo aussi nulle.

Ca blesse. Mais l'effet est comme une médecine amère. *Le prochain sprint*, l'équipe essaiera d'avoir des choses vraiment *terminées* ! Ils se diront que « bien, peut-être nous ne pouvons montrer que 2 choses au prochain sprint au lieu de 5, mais merde cette fois ça MARCHERA ! ». L'équipe sait qu'elle aura à faire une démo, peu importe quoi, ce qui augmente les chances qu'il y ait quelque chose d'utile à montrer. J'ai vu arriver cela plusieurs fois.

C'est crucial dans un contexte multi-équipes. Toutes les personnes impliquées ont besoin de voir le produit intégré ensemble de manière régulière. Il y aura toujours des problèmes d'intégration, mais plus tôt vous les découvrez, plus faciles ils seront à résoudre. L'auto-organisation fonctionne seulement avec la transparence et les boucles de feedback, et une revue de sprint correctement exécutée fournit les deux.

Checklist pour les démos de sprint

- Assurez-vous de présenter clairement l'objectif du sprint. S'il y a des personnes à la démo qui ne connaissent pas du tout votre produit, prenez quelques minutes pour le présenter.
- Ne dépensez pas trop de temps à préparer la démo, surtout pas pour une démo tape-à-l'œil. Ignorez ce qui ne marche pas et concentrez vous sur la démonstration du code qui marche.
- Gardez un rythme élevé, autrement dit essayez de préparer une démo rythmée plutôt qu'une jolie démo.
- Gardez la démo à un niveau métier, oubliez les détails techniques. Concentrez-vous sur « ce que nous avons fait » plutôt que sur « comment nous l'avons fait ».
- Si possible, laissez l'audience essayer elle-même le produit.
- Ne montrez pas les tas de corrections de bugs mineurs et de fonctionnalités insignifiantes. Mentionnez-les mais ne les montrez pas, car ça prend trop de temps et ça écarte l'attention des histoires plus importantes.

Certaines équipes font deux revues: une courte revue publique, ciblée aux parties prenantes externes, suivie d'une revue interne avec plus de détails et des choses

comme les challenges clés et les décisions techniques prises sur le chemin. Une excellente façon de diffuser le savoir entre équipes, et d'épargner les parties prenantes des détails techniques dont ils n'ont rien à faire.

S'occuper des choses « indémontrables »

Un membre de l'équipe : « Je ne vais pas présenter cet élément, parce que ce n'est pas démontrable. L'histoire est 'Améliorer la montée en charge pour que le système puisse supporter 10000 utilisateurs simultanément'. Je ne peux tout de même pas inviter 10000 utilisateurs à la démo ? »

Scrum master : « Cet élément est-il terminé ? »

Membre de l'équipe. « Oui, bien sûr ».

Scrum master: « Comment le sais-tu ? »

Membre de l'équipe : « J'ai installé le système sur un environnement de tests de performance, j'ai démarré 8 serveurs de charge et j'ai stressé le système avec des requêtes simultanées ».

Scrum master: « Mais as-tu une indication comme quoi le système peut supporter 10000 utilisateurs ? ».

Membre de l'équipe : « Oui. Les machines de test sont pourries, cependant elles ont supporté 50000 requêtes simultanées pendant mon test ».

Scrum master: « Comment le sais-tu ? »

Membre de l'équipe (frustré): « Ben j'ai ce rapport ! Vous pouvez voir par vous-même, il montre comment le test était configuré et combien de requêtes ont été lancées ! »

Scrum master: « Excellent ! Alors voilà ta « démo ». Montre juste le rapport et fais le passer dans l'audience. Mieux que rien, non ? ».

Membre de l'équipe : « Ah, c'est suffisant ? Mais c'est moche, il faudrait le rendre plus présentable.».

Scrum master: «OK, mais n'y passe pas trop de temps. Ca n'a pas besoin d'être joli, juste informatif. »

10

Comment nous faisons les rétrospectives de sprint

Pourquoi nous insistons pour que toutes les équipes fassent des rétrospectives

La chose la plus importante à propos des rétrospectives est de *s'assurer qu'elles aient lieu.*

Pour certaines raisons, les équipes ne semblent pas toujours enclin à faire des rétrospectives. Si on ne les aiguille pas délicatement, la plupart des équipes laissent tomber la rétrospective et passent directement au sprint suivant. Il s'agit peut-être d'un truc culturel en Suède, pas sûr.

Non, je l'ai vu dans beaucoup de pays, alors c'est juste la nature humaine. On veut toujours aller à la chose suivante. Ironiquement, plus vous êtes stressé, plus vous aurez envie de sauter la rétrospective. Mais plus vous êtes stressé, plus vous avez sérieusement besoin de la rétrospective ! Un peu comme « je suis tellement pressé à couper ces arbres que je n'ai pas le temps de m'arrêter et aiguiser ma scie ! » Elles n'ont pas besoin de durer si longtemps toutefois. Pour des sprints de deux semaines, time-boxez la rétrospective à une heure. Mais faites une rétrospective plus longue (demi-journée ou journée entière) tous les deux mois pour pouvoir traiter les problèmes plus épineux.

Cependant, tout le monde semble d'accord, les rétrospectives sont extrêmement utiles. En fait, je dirais que la rétrospective est la seconde partie la plus importante de Scrum (la première étant la réunion de planification de sprint) parce que c'est la *meilleure chance de vous améliorer* !

Exactement. Et c'est pourquoi la rétrospective est la chose *la-plus-importante* dans Scrum, pas la seconde !

Bien sûr, vous n'avez pas besoin d'une réunion de rétrospective pour avoir de bonnes idées, vous pouvez le faire dans votre baignoire à la maison ! Mais

l'équipe va-t-elle accepter votre idée ? Peut-être, mais la probabilité d'avoir l'adhésion de l'équipe est beaucoup plus importante si l'idée vient « de l'équipe », c'est-à-dire si elle vient pendant la rétrospective quand tout le monde est autorisé à contribuer et à discuter des idées.

Sans les rétrospectives vous trouverez des équipes qui reproduiront les mêmes erreurs, encore et encore.

Comment nous organisons les rétrospectives

Le format général varie un peu, mais généralement nous faisons quelque chose comme ça :

- Nous allouons 1 à 3 heures selon le volume de discussions qui est anticipé.
- Participants : le directeur de produit, l'équipe complète, et moi-même.
- Nous allons dans une pièce fermée, un coin avec un canapé confortable, un patio, ou une place du même style. Du moment que nous pouvons discuter sans être dérangés.
- Nous ne faisons pas les rétrospectives dans le bureau de l'équipe, car l'attention des membres a tendance à se relâcher.
- Quelqu'un est désigné comme secrétaire.
- Le Scrum master montre le sprint backlog et, avec l'aide de l'équipe, résume le sprint. Les évènements et décisions importantes, etc.
- Nous faisons «des tours de table». Chaque personne a une chance de dire, sans être interrompu, ce qu'elle pense être bien, ce qu'elle pense qui peut être mieux, et ce qu'elle pense qui devrait être différent pour le prochain sprint.
- Nous comparons la vélocité estimée et la vélocité effective. S'il y a une grosse différence nous essayons d'analyser pourquoi.
- Quand le temps est presque écoulé le Scrum master essaie de résumer les suggestions concrètes pour améliorer le prochain sprint.

Nos rétrospectives ne sont généralement pas très structurées. Le thème implicite est toujours le même : « que pouvons-nous améliorer au prochain sprint ».

Ceci est un exemple de tableau blanc lors d'une récente rétrospective :

Trois colonnes :

- **Bien** *:* Si nous pouvions refaire le même sprint, nous ferions ces choses exactement pareil.
- **Peut mieux faire** *:* Si nous pouvions refaire le même sprint, nous ferions ces choses différemment.
- **Améliorations** *:* Idées concrètes pour s'améliorer dans le futur.

Les colonnes 1 et 2 concernent le passé, tandis que la colonne 3 concerne le futur.

Après que les membres de l'équipe aient déballé tous ces post-its, ils utilisent « le vote par points » pour déterminer quelles améliorations prendre en compte au prochain sprint. Chaque membre de l'équipe dispose de 3 aimants et sont invités à voter pour les améliorations qu'ils aimeraient voir pris en compte en priorité pendant le prochain sprint. Chaque membre peut distribuer ces aimants comme il veut, il peut même placer les 3 sur le même post-it.

En se basant là-dessus ils sélectionnent les 5 améliorations à faire en priorité, et les suivront à la prochaine rétrospective.

C'est important de ne pas être trop ambitieux ici. Concentrez-vous sur un petit nombre d'améliorations par sprint.

Il y a plein de façons de faire des rétrospectives. Variez le format, pour que la réunion ne s'évente pas. Vous trouverez plein d'idées dans le livre *Agile Retrospectives*. Retromat, un générateur aléatoire de rétrospectives, est amusant, également (www.plans-for-retrospectives.com). :o)

Cependant, je remarque que je reviens souvent au format simple décrit ci-dessus. Il marche pour la majorité des cas. Ou même plus simplement, prenez une pause café de 20 minutes avec deux sujets de discussions : « Quoi garder » et « Quoi changer ». Un peu superficiel, mais mieux que rien !

La diffusion des enseignements tirés entre équipes

Les informations tirées d'une rétrospective de sprint ont généralement beaucoup de valeur. Est-ce que cette équipe a du mal à se concentrer parce que le directeur des ventes kidnappe les programmeurs en tant « qu'experts techniques » dans les réunions de vente. ? C'est une information importante. Peut-être que les autres équipes ont le même problème ? Devrions-nous mieux éduquer la direction à propos de nos produits, afin qu'ils puissent faire eux-mêmes le support des ventes ?

Ou mieux maintenant, invitez les manager des ventes à une réunion, informez vous de leurs besoins, et discutez des solutions possibles ensemble !

Une rétrospective ne se limite pas à comment cette équipe peut faire mieux le prochain sprint, ça a des implications plus larges.

Notre stratégie pour appréhender ça est très simple. Une personne (dans ce cas, moi) assiste à toutes les rétrospectives de sprint et agit comme un pont de connaissance. Assez simple.

Une alternative serait que chaque équipe Scrum publie un rapport de rétrospective de sprint. Nous avons essayé ça mais nous trouvons que peu de gens lisent les rapports, et encore moins s'y conforment. C'est pourquoi nous faisons ça de manière simple à la place.

Les règles importantes pour la personne qui fait le « pont de connaissances » :

- Il doit savoir écouter.
- Si la rétrospective est trop silencieuse, il doit être prêt à poser des questions simples mais ciblées afin de stimuler les discussions au sein du groupe. Par exemple « si vous pouviez remonter le temps et refaire le même sprint depuis le 1er jour, qu'est-ce que vous changeriez ? ».
- Il devrait prendre le temps d'aller aux rétrospectives de toutes les équipes.
- Il devrait avoir une certaine autorité, de façon à mettre en œuvre les suggestions d'amélioration qui ne sont pas sous le contrôle de l'équipe.

Cela marche assez bien mais il peut y avoir d'autres approches qui fonctionnent encore mieux. Dans ce cas, éclairez-moi.

Echanger les facilitateurs est un bon modèle. Comme « je faciliterais la rétrospective de ton équipe si tu facilites la mienne. » Cela fonctionne pour une diffusion de connaissance bidirectionnelle simple, et vous permet aussi en tant que Scrum master de participer entièrement à la rétrospective de votre équipe (plutôt que de faciliter).

Changer ou ne pas changer

Disons que l'équipe conclut que « nous ne communiquons pas assez dans l'équipe, nous nous marchons dessus et nous mettons la pagaille dans la conception des autres. »

Que feriez-vous pour y remédier ? Introduire des réunions quotidiennes sur la conception ? Introduire de nouveaux outils pour faciliter la communication ? Ajouter plus de pages dans le wiki ? C'est bien, peut-être. Mais encore une fois, peut-être pas.

Nous avons observé que, dans beaucoup de cas, identifier clairement un problème est suffisant pour qu'il se résolve tout seul au prochain sprint. Surtout si vous accrochez la rétrospective de sprint sur le mur du bureau de l'équipe (ce que nous oublions toujours de faire, honte à nous !). Chaque chose que vous introduisez a un coût, aussi avant d'introduire des changements, envisagez de ne rien faire du tout et espérez que le problème disparaîtra (ou réduira) automatiquement.

L'exemple ci-dessus (« nous ne communiquons pas assez dans l'équipe… ») est un exemple typique où la meilleure solution est de ne rien faire.

Si vous introduisez un nouveau changement à chaque fois quelqu'un se plaindra, les gens seront réticents à remonter les problèmes mineurs, ce qui serait terrible.

Exemples de choses qui peuvent remonter pendant les rétrospectives

Voici quelques exemples typiques de ce qui peut arriver pendant un sprint planning, et les actions typiques.

« Nous devrions passer plus de temps à décomposer les histoires en sous-éléments et tâches »

C'est assez courant. Chaque jour lors de la mêlée quotidienne, des membres de l'équipe se retrouvent à dire « Je ne sais pas trop quoi faire aujourd'hui ». Alors

après chaque mêlée quotidienne vous passez du temps à trouver des tâches concrètes. C'est généralement plus efficace de faire ça en amont.

Actions typiques *:* aucune. L'équipe règlera ça probablement d'elle-même à la prochaine réunion de planification de sprint. Si cela se répète, prévoyez plus de temps pour le sprint planning.

« Trop de dérangements extérieurs »

Actions typiques :

• demandez à l'équipe de réduire leur facteur de focalisation pour le prochain sprint, afin d'avoir un plan plus réaliste

• demandez à l'équipe de bien noter les dérangements pendant le prochain sprint. Qui les dérange, combien de temps. Ca aidera à résoudre le problème plus tard.

• demandez à l'équipe d'essayer de rediriger tous les dérangements vers le Scrum master ou le directeur de produit

• demandez à l'équipe de désigner une personne comme « gardien », tous les dérangements sont redirigés vers lui, ainsi le reste de l'équipe peut se concentrer. Ce peut être le Scrum master ou quelqu'un à tour de rôle.

Le modèle du gardien de but tournant est extrêmement commun et marche généralement bien. Essayez-le !

« Nous nous sommes trop engagés et nous n'avons terminé que la moitié du travail »

Actions typiques : aucune. L'équipe ne s'engagera probablement pas trop au prochain sprint. Ou du moins pas autant que cette fois.

Au fait, dès 2014, le terme "engagement de sprint" a complètement disparu du Guide Scrum. A la place, il a été renommé « prévision de sprint ». Bien mieux ! Le mot « engagement » a causé tellement d'incompréhension. De nombreuses équipes pensaient que le plan de sprint était une sorte de promesse (un peu ridicule, en considérant que l'une des quatre valeurs fondamentales en agile est « l'adaptation au changement plus que le suivi d'un plan »). Le plan de sprint n'est pas un engagement, c'est une prévision et une hypothèse – « C'est la manière dont nous pensons pouvoir au mieux atteindre l'objectif de sprint. »

Néanmoins, c'est vraiment nul de livrer constamment moins que prévu. Si c'est un problème, commencez par strictement appliquer la météo de la veille, et tirez seulement le nombre de points d'histoire que vous avez terminés au dernier sprint (ou la moyenne des trois derniers sprints si vous voulez faire un peu de zèle). Cette astuce simple et puissante fait généralement fondre le problème vu que la vélocité devient auto-ajustée.

« Notre bureau est trop bruyant et bordélique »

Actions typiques :

- essayez de créer un meilleur environnement, ou déménagez l'équipe. Louez une chambre d'hôtel. Ce que vous voulez. Voir page 68 « Comment nous organisons le bureau de l'équipe »).
- Si ce n'est pas possible, dîtes à l'équipe de réduire leur facteur de focalisation, et indiquez clairement que c'est à cause du bruit et du désordre. En espérant que le directeur de produit commencera à harceler la direction à ce sujet.

Heureusement je n'ai jamais eu à menacer de déménager l'équipe. Mais je le ferai si je le devais :o)

11

Relâcher le rythme entre les sprints

Dans la vraie vie, vous ne pouvez pas sprinter tout le temps. Vous avez besoin de vous reposer entre deux sprints. Si vous sprintez tout le temps, vous êtes en effet juste en train de faire un jogging.

Cela s'applique à la vie également, pas juste Scrum ! Par exemple, si je n'avais pas de jeu dans ma vie, ce livre n'aurait jamais existé (ni cette seconde édition, ou n'importe lequel de mes autres livres, articles, vidéos, etc.). Le jeu est super important pour à la fois la productivité et le bien être personnel ! Si vous êtes l'une de ces personnes avec un *calendrier-toujours-plein*, essayez cela : ouvrez votre calendrier et bloquez une demi-journée par semaine, écrivez « jeu » ou « non réservable » ou quelque chose. Ne décidez pas à l'avance de ce que vous ferez à ce moment, voyez juste ce qu'il se passe. :o)

C'est la même chose avec Scrum et le développement logiciel en général. Les sprints sont plutôt intenses. En tant que développeur vous ne vous relâchez jamais, chaque jour vous devez assister à cette fichue réunion et dire à tout le monde ce que vous avez fait la veille. Peu auraient tendance à dire «J'ai passé la majeure partie de la journée à mon bureau à surfer sur des blogs et siroter du capuccino».

En plus de la vraie pause elle-même, il y a une autre bonne raison d'avoir un relâchement entre les sprints. Après la démo du sprint et la rétrospective, à la fois l'équipe et le directeur de produit seront saturés d'informations et d'idées à digérer. S'ils se remettent immédiatement à courir et commencer à planifier le sprint suivant, il est peu probable que quiconque ait l'occasion de digérer une information ou des leçons apprises, que le directeur de produit n'aura pas le temps d'ajuster ses priorités après la démo de sprint, etc.

Mauvais :

Lundi
09-10: Démo sprint 1 10-11: Rétrospective sprint 1 13-16: Planning sprint 2

Nous essayons d'insérer du relâchement avant de commencer un nouveau sprint (plus spécifiquement, la durée *après* la rétrospective de sprint et *avant* la prochaine réunion de planification de sprint).

Tout au moins, nous essayons de faire en sorte que la rétrospective de sprint et la réunion de planification du sprint à venir n'arrivent pas le même jour. Tout le monde devrait avoir au moins une bonne nuit de sommeil « sans sprint » avant de démarrer un nouveau sprint.

Mieux :

Lundi	Mardi
09-10: Démo sprint 1 10-11: Rétrospective sprint 1	9-13: Planning sprint 2

Encore mieux :

Vendredi	Samedi	Dimanche	Lundi
09-10: Démo sprint 1 10-11: Rétrospective sprint 2			9-13: Planning sprint 2

Une façon de faire cela sont les «jours labo» (ou quoique vous choisissiez de les appeler). Ce sont des journées où les développeurs sont autorisés à faire essentiellement ce qu'ils veulent (OK, je le reconnais, inspiré par Google). Par exemple, étudier à fond les derniers outils et APIs, suivre une formation, discuter de trucs d'informatique avec les collègues, coder un projet perso, etc.

Notre but est d'avoir un jour labo entre chaque sprint. De cette façon vous avez naturellement une pause entre les sprints, et vous avez une équipe de développement qui a de réelles chances de garder à jour leurs connaissances. De plus, c'est un avantage plutôt attractif pour l'employeur.

Meilleur?

Jeudi	Vendredi	Samedi	Dimanche	Lundi
09-10: Démo sprint 1 10-11: Rétrospective sprint 1	JOURNÉE LABO			9-13: Planning sprint 2

Actuellement nous avons des «jours labo» une fois par mois. Le premier vendredi de chaque mois pour être précis. Pourquoi pas entre les sprints à la place ? Eh bien, parce que j'ai pensé que c'était important que toute l'entreprise soit en journée labo en même temps. Autrement les gens ont tendance à ne pas le faire sérieusement. Et comme nous (jusque là) n'avons pas aligné les sprints à travers tous les produits, j'ai du choisir à la place une journée labo indépendamment des sprints.

On pourrait un jour tenter de synchroniser les sprints à travers tous les produits (i.e. même démarrage et fin de sprint pour tous les produits et les équipes). Dans ce cas nous mettrons définitivement un jour labo entre chaque sprint.

Après de nombreuses expérimentations à Spotify, nous avons fini par faire des semaines de piratage à l'échelle de l'entreprise. Deux fois par an, nous faisons une semaine complète de faites-ce-que-vous-voulez, avec une démo et une fête le vendredi. Toute l'entreprise, pas juste les techos. La quantité d'innovations que cela à déclenché est juste incroyable ! Et parce que tout le monde le fait en même temps, les équipes sont moins susceptibles d'être perturbées par des dépendances. J'ai fait une vidéo sur la culture d'ingénierie Spotify qui décrit la semaine de piratage et plein d'autres choses. Jetez-y un œil sur http://tinyurl.com/spotifyagile

12

Comment nous faisons la planification de release et les contrats au forfait

Dès fois, on a besoin de planifier à l'avance plus d'un sprint à la fois. Typiquement dans un contexte de contrat au forfait où on *doit* planifier en avance, ou sinon on risque de signer pour quelque chose qu'on ne peut délivrer à temps.

Typiquement, la planification de release est pour nous une tentative de répondre à la question « *quand*, au *plus tard*, serons-nous en mesure de délivrer la version 1.0 de ce nouveau système ? ».

Si vous voulez *vraiment* apprendre la planification de release je vous suggère de sauter ce chapitre et à la place d'acheter le livre de Mike Cohn « Agile estimating and planning ». J'aurais voulu avoir lu ce livre plus tôt (Je l'ai lu *après* que nous soyons arrivés à comprendre par nous mêmes…). Ma version de la planification de release est assez simpliste mais elle devrait être un bon point de départ.

Aussi regardez le livre Lean Startup, d'Eric Ries. Un gros problème est que la plupart des projets tendent à développer une release big-bang, au lieu de livrer de petits incréments et de mesurer pour voir s'ils sont sur le bon chemin. Le lean startup, si appliqué convenablement, réduit radicalement le risque et le coût de l'échec.

Définir vos seuils d'acceptation

En plus du backlog de produit habituel, le directeur de produit définit une liste des seuils d'acceptation qui est une classification simple de ce que signifient les niveaux d'importance utilisés dans le backlog de produit par rapport aux aspects contractuels.

Voici un exemple de règles de seuils d'acceptation :

- Tous les éléments avec une importance >= 100 *doivent* être inclus dans la version 1.0, ou sinon nous serions condamnés à payer des pénalités de retard.
- Tous les éléments avec une importance de 50-99 devraient être inclus dans la version 1.0, mais nous devrions être en mesure de nous en sortir en les incluant dans une release disponible à court terme.
- Les éléments avec une importance de 25-49 sont requis mais peuvent être faits dans une version future 1.1.
- Les éléments avec une importance <25 sont spéculations et pourraient ne jamais être requis.

Et voici un exemple de backlog de produit, avec un code couleur basé sur les règles ci-dessus.

Importance	Nom
130	banane
120	pomme
115	orange
110	goyave
100	poire
95	raisin
80	cacahuète
70	beignet
60	oignon
40	pamplemousse
35	papaye
10	myrtille
10	pêche

Rouge = *doit* être inclus dans la version 1.0 (banane – poire)
Jaune = *devrait* être inclus dans la version 1.0 (raisin – oignon)
Vert = peut être fait plus tard (pamplemousse – pêche)

Donc si nous livrons tous les éléments de banane à oignon à temps nous sommes sains et saufs. Si le temps se met à manquer nous *devrions* nous en sortir en écartant raisin, cacahuète, beignet ou oignon. Tout ce qui est en dessous d'oignon est du bonus.

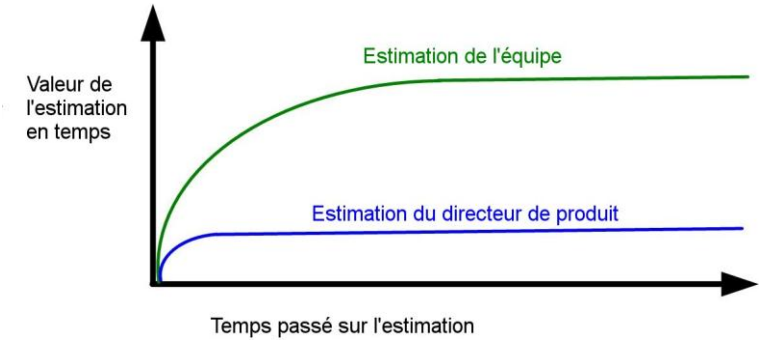
Et vous pouvez, bien sûr, faire cette analyse sans avoir d'évaluation d'importance numérique ! Ordonnez juste la liste. Mais vous l'avez déjà compris.

Estimation en temps des éléments les plus importants

Afin de faire la planification de releases, le directeur de produit a besoin d'estimations, au moins pour toutes les histoires qui sont incluses dans le contrat. Tout comme la planification de sprint, il s'agit d'un effort mutuel entre le directeur de produit et l'équipe – l'équipe estime, le directeur de produit décrit les éléments et répond aux questions.

Une estimation en temps a de la valeur si elle se révèle être proche de la réalité, moins de valeur si elle se révèle être à coté, disons, de 30%, et complètement sans valeur si elle n'a aucune relation avec la réalité.

Voici mon interprétation de la valeur d'une estimation en temps en relation avec ceux qui la calculent et combien de temps ils y passent dessus.



Tout cela est juste une façon verbeuse de dire :

- Laissez l'*équipe* faire les estimations.
- Ne leur faites pas passer trop de temps dessus.
- Assurez-vous qu'ils comprennent que les estimations en temps sont des *estimations approximatives*, non des *engagements*.

Habituellement le directeur de produit rassemble toute l'équipe dans une pièce, fournit quelques boissons, et lui explique que le but de cette réunion est d'estimer en temps les 20 premières (ou autre) histoires du backlog de produit. Il aborde chaque histoire une fois, puis il laisse l'équipe retourner travailler. Le directeur de produit reste dans la pièce pour répondre aux questions et clarifier le périmètre de chaque histoire si nécessaire. Juste comme pendant la planification du sprint, le champ « comment démontrer » est un moyen très utile pour réduire le risque de malentendus.

Cette réunion doit être strictement bornée en temps, sans quoi les équipes auraient tendance à passer trop de temps à estimer trop peu d'histoires.
Si le directeur de produit veut passer plus temps sur cela, il programme simplement une autre réunion plus tard. L'équipe doit s'assurer que l'impact de ces réunions sur leurs sprints en cours est clairement visible pour le directeur produit, de telle sorte qu'il comprenne que leurs estimations en temps ne sont pas gratis.

Voici un exemple qui montre à quoi les estimations en temps pourraient ressembler (en points d'histoire) :

Imp	Nom	Estimation
130	banane	12
120	pomme	9
115	orange	20
110	goyave	8
100	poire	20
95	raisin	12
80	cacahuète	10
70	beignet	8
60	oignon	10
40	pamplemousse	14
35	papaye	4
10	myrtille	
10	pêche	

Très déroutant de les appeler "estimations en temps". Appelez les « estimations en taille » à la place. Je ne sais pas combien de temps « banane » va prendre, mais je suis plutôt sûr que ce sera un peu plus long que « pomme » et beaucoup plus court que « orange ». Mieux vaut être grossièrement juste que précisément faux !

OK, alors maintenant nous avons des sortes d'estimations brutes pour les histoires les plus importantes. La prochaine étape est d'estimer notre vélocité moyenne par sprint.

Cela veut dire que nous avons besoin de définir notre facteur de focalisation. Voir page XXX « Comment l'équipe choisit les histoires à inclure dans le sprint ? ».

Oh non, pas ce fichu facteur de focalisation encore…

Le facteur de focalisation revient fondamentalement à se demander « quelle est la part du temps passé par l'équipe consacrée aux histoires adoptées en cours ? ». Ce n'est jamais 100% car les équipes perdent du temps en faisant des choses non prévues, en changeant de contexte, en aidant les autres équipes, en lisant leurs emails, en réparant les problèmes d'ordinateur, en discutant politique dans la cuisine, etc.

Disons que nous déterminons un facteur de focalisation de 50% pour l'équipe (très bas, normalement on tourne autour de 70%). Et disons que notre sprint dure 3 semaines (15 jours) et que la taille de l'équipe et 6.

Chaque sprint fait alors 90 jours*homme, mais produira seulement 45 jours*homme d'histoires (à cause du facteur de focalisation de 50%).

Notre vélocité estimée est donc de 45 points d'histoire.

Ignorez le facteur de focalisation. Demandez à l'équipe de regarder fixement la liste et de faire une estimation éclairée sur jusqu'où ils peuvent aller en un sprint. Additionnez les points. Cela sera plus rapide que le facteur de focalisation, et à peu près aussi précis/imprécis. Mieux vaut être grossièrement juste que… – oh, attendez. Je l'ai déjà dit.

Si chaque histoire avait une estimation en temps de 5 jours (ce qui n'est pas le cas) alors cette équipe avalerait approximativement 9 histoires par sprint.

Tout mettre ensemble dans un plan de release

Maintenant que nous avons des estimations en temps et une vélocité (45) nous pouvons facilement découper le backlog de produit en plusieurs sprints :

Imp	Nom	Estimation
		Sprint 1
130	banane	12
120	pomme	9
115	orange	20
		Sprint 2
110	goyave	8
100	poire	20
95	raisin	12
		Sprint 3
80	cacahuète	10
70	beignet	8
60	oignon	10
40	pamplemousse	14
		Sprint 4
35	papaye	4
10	myrtille	
10	pêche	

Chaque sprint contient autant d'histoires que possible dans la limite de la vélocité estimée de 45.

Maintenant on peut voir que nous aurons probablement besoin de 3 sprints pour finir tous les « *doit* être inclus » et les « *devrait* être inclus ».

3 sprints = 9 semaines calendaires = 2 mois calendaires. Maintenant est-ce vraiment la date de livraison promise au client ? Cela dépend entièrement de la nature du contrat ; à quel point le périmètre est-il fixé, etc. Généralement nous ajoutons une réserve significative pour nous protéger contre les mauvaises estimations en temps, les problèmes imprévus, les fonctionnalités inattendues, etc. Donc dans ce cas nous pouvons nous mettre d'accord sur une livraison dans 3 mois, ce qui nous donne 1 mois de « réserve ».

Ce qui est bien c'est que nous pouvons démontrer quelque chose d'utilisable au client toutes les 3 semaines et l'inviter à modifier ses exigences au fur à mesure que nous avançons (cela dépend bien sûr du type de contrat).

Adapter le plan de release

La réalité ne s'adaptera pas à un plan, c'est plutôt le cas inverse qui se passe.

Après chaque sprint, nous regardons la vélocité effective pour ce sprint. Si la vélocité effective était très différente de la vélocité estimée, nous revoyons la vélocité estimée pour les futurs sprints et mettons à jour le plan de release. Si cela devait nous mettre en situation difficile, le directeur de produit pourrait commencer soit à négocier avec le client, soit à voir comment il peut réduire le périmètre sans rompre le contrat. Ou peut-être lui et l'équipe trouvent le moyen d'augmenter la vélocité ou le facteur de focalisation en supprimant des obstacles sérieux qui ont été identifiés durant le sprint.

Le directeur de produit pourrait appeler le client et dire « Bonjour, nous sommes un peu en retard mais je crois que nous pouvons livrer à temps si nous enlevons la fonctionnalité «Pacman embarqué» qui prend beaucoup de temps à développer. Nous pourrons l'inclure dans une version qui sortira 3 semaines après la première release si vous le souhaitez ».

Ce n'est peut-être pas une bonne nouvelle pour le client, mais au moins nous sommes honnêtes et nous laissons le choix au client tôt dans le projet – doit-on livrer les choses les plus importantes à temps ou doit-on tout livrer après la date prévue ? Ce n'est pas un choix si difficile en général :o)

J'ai fait une vidéo de 15 minutes qui s'appelle "Agile Product Ownership in a Nutshell". Elle contient des trucs et astuces utiles autour du planning de release et de la gestion de backlog en Scrum. Jetez-y un œil !
http://tinyurl.com/ponutshell

13

Comment nous combinons Scrum avec XP

Dire que Scrum et XP (eXtreme Programming) peuvent être combinés avec profit n'est pas une déclaration réellement sujette à controverse. La plupart des choses que je vois sur le net sont en faveur de cette hypothèse, donc je ne vais pas passer de temps à argumenter pourquoi.

Tout de même, je vais mentionner une chose. Scrum se concentre sur le management et les pratiques d'organisation tandis que XP se concentre surtout sur les pratiques de programmation concrètes. C'est pour ça qu'ils fonctionnent bien ensemble – ils concernent différentes zones et sont complémentaires.

Je déclare donc formellement que j'ajoute ma voix aux preuves empiriques déjà existantes que Scrum et XP peuvent être combinés de manière productive !

J'ai appris de Jeff Sutherland que le premier Scrum avait en fait toutes les pratiques XP. Mais Ken Schwaber l'a convaincu de laisser les pratiques d'ingénierie en dehors de Scrum, pour garder le modèle simple et laisser les équipes prendre la responsabilité elles-mêmes pour les pratiques techniques. Peut-être que cela a aidé à diffuser Scrum plus rapidement, mais le bémol est que beaucoup d'équipe souffrent parce qu'elles manquent de pratiques techniques qui leur permettraient de rendre possible un développement agile soutenable.

Je vais souligner certaines des pratiques XP les plus intéressantes, et comment elles s'appliquent à notre travail de tous les jours. Toutes nos équipes n'ont pas réussi à adopter toutes les pratiques, mais au total nous avons expérimenté la plupart des aspects de la combinaison XP/Scrum. Certaines pratiques XP sont directement prises en compte par Scrum et peuvent être vues comme du recouvrement, par exemple « Toute l'équipe », « S'asseoir ensemble », « Histoires » et « Jeu du planning ». Dans ces cas nous nous en sommes simplement tenus à Scrum.

Nous avons commencé cela récemment dans une de nos équipes. Ca a plutôt bien fonctionné en fait. La plupart de nos autres équipes ne travaillent toujours pas beaucoup en duo, mais après l'avoir réellement essayé avec une équipe pendant maintenant plusieurs sprints, je suis motivé pour essayer d'entraîner plus d'équipes à faire l'essai.

Quelques conclusions pour l'instant sur la programmation en binôme :

- La programmation en binôme améliore la qualité du code.
- La programmation en binôme améliore la focalisation de l'équipe (par exemple quand le gars derrière vous dit « hé est-ce que ce truc est vraiment nécessaire pour ce sprint ? »).
- Curieusement beaucoup de développeurs qui sont fortement opposés à la programmation en binôme ne l'ont pas réellement essayée, et apprennent rapidement à l'apprécier une fois qu'ils l'essayent.
- La programmation en binôme est épuisante et ne devrait pas être pratiquée toute la journée.
- Faire tourner fréquemment les duos est bénéfique.
- La programmation en binôme améliore la diffusion des connaissances dans le groupe. De façon étonnamment rapide.
- Quelques personnes ne sont simplement pas à l'aise avec la programmation en binôme. Ne jetez pas un excellent programmeur juste parce qu'il n'est pas à l'aise avec la programmation en binôme.
- Les revues de code sont une alternative acceptable à la programmation en binôme.
- Le « copilote » (le type qui n'utilise pas le clavier) devrait également avoir un ordinateur à lui. Pas pour le développement, mais pour de petites activités quand c'est nécessaire, pour parcourir la documentation quand le « pilote » (le type au clavier) est bloqué, etc.
- N'imposez pas la programmation en binôme aux gens. Encouragez-les et fournissez les bons outils mais laissez-les expérimenter à leur propre rythme.

Amen ! Ceci, pour moi, est plus important que Scrum et XP tous les deux. Vous pouvez prendre ma maison et ma télé et mon chien, mais n'essayez pas de m'empêcher de faire du DDT ! Si vous n'aimez pas le DDT alors ne me laissez pas entrer dans votre bâtiment, parce que je vais essayer d'en introduire d'une manière ou d'une autre :o)

OK, je ne suis plus aussi religieux à ce sujet. J'ai réalisé que le DDT était une technique de niche que très peu de personnes avaient la patience de maîtriser. A la place, j'enseigne les techniques et puis je laisse les équipes décider de la mesure de son utilisation, et du moment.

Voici un résumé du DDT en 10 secondes:

> *Le développement dirigé par les tests signifie que vous écrivez un test automatisé, puis vous écrivez juste assez de code pour faire passer ce test, puis vous remaniez le code principalement pour améliorer la lisibilité et supprimer les duplications. Rincez et recommencez.*

Quelques réflexions sur le développement dirigé par les tests.

- Le DDT est *difficile*. Cela prend du temps à un programmeur pour *voir la lumière*. En fait, dans beaucoup de cas, le temps que vous passez à former et à démontrer n'a pas réellement beaucoup d'importance – dans beaucoup de cas la seule manière pour un programmeur de *voir la lumière* est de le faire programmer en duo avec quelqu'un de vraiment bon en DDT. Une fois qu'un programmeur *voit la lumière*, cependant, il sera en principe gravement infecté et ne voudra jamais plus travailler d'une autre manière.
- Le DDT a un effet profondément positif sur la conception du système.
- Cela prend du temps pour avoir du DDT parfaitement au point dans un nouveau produit, surtout des tests d'intégration en boite noire, mais le retour sur investissement est *rapide*.
- Assurez-vous d'investir le temps nécessaire pour que ce soit *facile* d'écrire des tests. Cela signifie obtenir les bons outils, éduquer les gens, fournir des classes utilitaires ou des classes de base appropriées, etc.

Etant donné que le DDT est si difficile, je n'essaie pas de forcer les personnes, à la place je coache ces principes :

1) Assurez-vous que chaque fonctionnalité clé a au moins un test d'acceptation de bout-en-bout, interagissant par l'IGU (Interface Graphique Utilisateur) ou juste derrière.

2) Assurez-vous que tout code complexe ou critique pour le business est couvert par des tests unitaires

3) Cela va laisser du code non couvert. Ce n'est pas grave. Mais soyez au courant du code qui n'est pas couvert ; assurez-vous que c'est un compromis délibéré plutôt que juste une négligence.

4) Ecrivez les tests au fur et à mesure, ne les gardez pas pour plus tard (vous serez tout aussi occupé plus tard que maintenant).

Cela semble donner suffisamment de durabilité sans avoir à recourir à un DDT complet. La couverture de tests finit généralement autour des 70% dû à la loi des rendements décroissants. En bref, l'automatisation des tests est cruciale, mais le DDT est optionnel.

Nous utilisons les outils suivants pour le développement dirigé par les tests :

- jUnit / httpUnit / jWebUnit. Nous envisageons TestNG et Selenium.
- HSQLDB en tant que BD embarquée en mémoire pour les tests.
- Jetty en tant que container web embarqué en mémoire pour les tests.
- Cobertura pour les métriques de couverture de test.
- Spring pour câbler différents types de montages de test (avec mocks, sans mocks, avec base de données externe, avec base de données en mémoire, etc).

Dans nos produits les plus sophistiqués (du point de vue du DDT) nous avons des tests d'acceptation automatisés de type boîte noire. Ces tests démarrent le système complet en mémoire, y compris les bases de données et les serveurs web, et accèdent au système en utilisant uniquement ses interfaces publiques (par exemple HTTP).

Cette façon de travailler est plus facile qu'avant, grâce à toute la ribambelle d'outils et de cadres de tests disponibles. Très utiles, que vous fassiez du DDT ou non.

Cela permet des cycles développement-construction-test extrêmement rapides. Cela agit également comme un filet de sécurité, donnant suffisamment de confiance aux développeurs pour remanier (*refactor*) souvent, ce qui signifie que la conception reste propre et simple même quand le système grandit.

Le DDT sur du nouveau code

Nous pratiquons le DDT pour tous les nouveaux développements, même si cela signifie que la mise en place initiale du projet prend plus de temps (puisqu'il nous faut plus d'outils et de support pour les harnais de test, etc). Il n'y a pas vraiment de question à se poser, les bénéfices sont tellement importants qu'il n'y a vraiment pas d'excuse pour *ne pas* faire du DDT.

Le DDT sur du vieux code

Le DDT est difficile, mais faire du DDT sur une base de code qui n'a pas été construite avec du DDT dès le début… ça c'est *vraiment difficile* ! Pourquoi ? Eh bien, en fait, je pourrais écrire beaucoup de pages sur ce sujet, alors je pense que je vais m'arrêter ici. Je garde cela pour mon prochain papier « le DDT depuis les tranchées » :o)

Jamais eu l'occasion de l'écrire…. Il ne manque pas de livres autour du TDD toutefois ! Et un excellent est au sujet du code legacy s'intitulant *Working Effectively with Legacy Code*, de Michael Feathers, un vrai classique. J'ai également écrit quelques articles sur la dette technique, regardez mon blog.
http://blog.crisp.se/tag/technical-debt

Nous avons passé pas mal de temps à essayer d'automatiser les tests d'intégration de l'un de nos systèmes les plus complexes, une base de code qui existe depuis un moment et qui était dans un état de pagaille extrême et était complètement dépourvue de tests.

A chaque release du système nous avions une équipe de testeurs dédiés qui réalisaient un lot complet de tests de régression et de performance. Les tests de régression étaient essentiellement manuels. Ceci ralentissait notablement notre cycle de développement et release. Notre but était d'automatiser ces tests. Après nous être cogné la tête contre le mur pendant plusieurs mois, toutefois, nous n'avons pas beaucoup avancé.

Après ça nous avons changé d'approche. Nous avons accepté le fait que nous étions obligés de vivre avec du test de régression manuel, et nous avons plutôt commencé à nous demander « comment pouvons-nous rendre le processus de test manuel moins consommateur de temps ? » C'était un système de jeu, et nous avons réalisé qu'une bonne partie du temps de l'équipe de test était consacrée à des tâches de mise en place assez triviales, comme bricoler dans le back office pour mettre en place des tournois pour le test, ou attendre qu'un tournoi planifié démarre. Donc nous avons créé des utilitaires pour cela. Des raccourcis et des scripts petits, facilement accessibles qui font tout le travail de base et permettent aux testeurs de se concentrer sur le véritable test.

Cet effort a vraiment été rentable ! En fait, c'est probablement ce que nous aurions dû faire au départ. Nous étions tellement impatients d'automatiser le test que nous avons oublié de le faire étape par étape, la première étape étant de construire des choses qui rendent le test *manuel* plus efficace.

Leçon tirée : si vous êtes coincés avec du test de régression manuel, et voulez vous en débarrasser en l'automatisant, ne l'automatisez pas (à moins que ce soit vraiment facile). A la place, construisez des choses qui rendent le test de régression manuel plus facile. *Ensuite*, considérez l'automatisation du véritable test.

La conception incrémentale

Cela signifie garder la conception simple au départ et l'améliorer continuellement, plutôt qu'essayer de tout faire parfaitement dès le départ et ensuite tout geler.

Nous sommes plutôt bons à ça, c'est-à-dire que nous passons un temps raisonnable à remanier et améliorer la conception existante, et que nous passons rarement du temps à faire de grandes conceptions à l'avance. Quelques fois nous échouons bien sûr, par exemple en permettant à une conception branlante de s'implanter trop fortement, si bien que le remaniement devient un gros projet. Mais tout bien considéré nous sommes raisonnablement satisfaits.

L'amélioration continue de la conception est principalement un effet de bord automatique du DDT.

L'intégration continue

La plupart de nos produits ont un système d'intégration continue plutôt sophistiqué, basé sur Maven et QuickBuild. Cela est très efficace et fait économiser du temps. C'est la solution ultime au bon vieux problème « hé mais ça marche sur ma machine ». Notre serveur de build en continu sert de « juge » ou point de référence à partir duquel on peut déterminer la santé de toutes nos bases de code.

Chaque fois que quelqu'un enregistre quelque chose dans le système de gestion de version, le serveur de build en continu reconstruit tout à partir de zéro sur un serveur partagé, et exécute tous les tests. Si quelque chose ne va pas il envoie un email notifiant toute l'équipe que le build a échoué, en indiquant quel changement dans le code a cassé le build, des liens sur les rapports de test, etc.
Chaque nuit le serveur de build en continu va reconstruire le produit à partir de zéro et va publier les binaires (fichiers de type .EAR, .WAR, etc.), la documentation, les rapports de test, les rapports de couverture de test, les rapports de dépendance, etc, sur notre portail interne de documentation. Certains produits vont aussi être déployés automatiquement sur un environnement de test.

Mettre en place tout cela a représenté *beaucoup de travail*, mais chaque minute passée en valait la peine.

Si vous poussez cela juste un peu plus loin, vous avez de la livraison continue. Chaque commit est un candidat de release, et effectuer une release est une opération en un seul clic. Je vois de plus en plus d'équipe le faire, et je le fais pour tous mes projets personnels. C'est incroyablement efficace et cool ! Je vous suggère de lire (ou au moins parcourir) le livre *Continuous Delivery*. Sa mise en place demande beaucoup de travail, mais en vaut vraiment la peine en début de tout nouveau produit. Rentable presque immédiatement. Et il y a d'excellents supports d'outils de nos jours.

La propriété collective du code

Nous encourageons la propriété collective du code mais toutes les équipes ne l'ont pas encore adoptée. Nous avons découvert que la programmation en binôme avec rotations fréquentes des duos conduit automatiquement à un niveau élevé de propriété collective du code. Les équipes avec ce niveau élevé de propriété collective ont prouvé qu'elles étaient très robustes, par exemple leur sprint ne meurt pas juste parce qu'une personne clé est malade.

Spotify (et bien d'autres sociétés dynamiques) a un model "interne open-source". Tout le code vit dans un GitHub interne et les gens peuvent cloner les repos et effectuer des pull requests comme tout autre projet public open-source. Très commode.

Un espace de travail informatif

Toutes les équipes ont accès à des tableaux blancs et de l'espace sur des murs vides, et en font un assez bon usage. Dans la plupart des pièces vous trouverez les murs couverts de toutes sortes d'informations sur le produit et le projet. Le plus gros problème est les vieux déchets qui s'accumulent sur les murs, il faudra peut-être introduire un rôle de « nettoyeur » dans chaque équipe.

Nous encourageons l'utilisation de tableaux de tâches, mais toutes les équipes ne les ont pas encore adoptés. Voir page 68 « comment nous organisons le bureau de l'équipe. »

Les normes de codage

Récemment nous avons commencé à définir des normes de codage. Très utiles, nous aurions dû le faire plus tôt. Cela ne prend presque pas de temps du tout, il faut juste commencer simple et les laisser se développer. Ecrivez juste les choses qui ne sont pas évidentes pour tout le monde et reliez-les à des supports existants chaque fois que c'est possible.

La plupart des programmeurs ont leur propre style de codage bien distinct. De petits détails comme comment ils gèrent les exceptions, comment ils commentent le code, quand ils retournent null, etc. Dans certains cas la différence n'a pas d'importance, dans d'autres cas elle peut conduire à une conception système gravement incohérente et du code très difficile à lire. Des normes de codage sont très utiles dans ce cas, du moins tant que vous vous concentrez sur les choses qui importent.

Voici quelques exemples de nos normes :

- Vous pouvez violer n'importe laquelle de ces règles, mais assurez-vous qu'il y a une bonne raison et documentez-là.
- Utilisez les conventions de codage de Sun par défaut : http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- Ne jamais, jamais, jamais, attraper des exceptions sans les relancer ou enregistrer la pile d'appel dans un journal. log.debug() c'est bien, mais simplement ne perdez pas cette pile d'appel.
- Utilisez l'injection de dépendances basée sur les accesseurs pour découpler les classes les unes des autres (sauf bien sûr quand un couplage fort est désiré).
- Evitez les abréviations. Les abréviations bien connues comme ADO sont acceptables.
- Les méthodes qui retournent des collections ou des tableaux ne devraient pas retourner null. Retournez des collections et des tableaux vides à la place de null.

Les standards de code et les guides de styles sont excellents. Mais il n'y a pas besoin de réinventer la roue – vous pouvez copier celui là de mon ami Google : http://google-styleguide.googlecode.com

Le rythme soutenable / le travail énergisé

Beaucoup de livres sur le développement logiciel agile prétendent que le dépassement des horaires est contre-productif dans le développement logiciel.

Après quelques expériences non désirées sur la question, je ne peux qu'être d'accord de tout cœur !

Il y a à peu près un an une de nos équipes (la plus grosse) faisait une quantité démente d'heures supplémentaires. La qualité de la base de code existante était déprimante et ils devaient passer la plupart de leur temps à éteindre les incendies. L'équipe de test (qui faisait aussi des heures supplémentaires) n'avait aucune chance de faire sérieusement le travail d'assurance qualité. Nos utilisateurs étaient mécontents et les tabloïdes nous mangeaient vivants.

Après quelques mois nous avons réussi à ramener le nombre d'heures à des niveaux décents. Les gens travaillent le nombre normal d'heures (sauf quelquefois dans des situations critiques). Et, surprise, la productivité et la qualité se sont améliorées de manière notable.

Bien sûr, la réduction des heures de travail n'a été en aucune manière le seul aspect ayant conduit à cette amélioration, mais nous sommes tous convaincus qu'elle a joué un grand rôle.

Je le vois encore et encore. En développement logiciel (et tout autre travail complexe, créatif), il y a très peu de corrélation entre les heures passées et la valeur produite. Ce qui compte sont les heures motivées et focalisées. La plupart d'entre-nous avons expérimenté cette sensation de venir un samedi matin, de travailler en paix et en silence pendant quelques heures, et d'avoir l'impression de terminer une semaine entière de travail dans ce temps ! C'est ce que je veux dire par heures motivées et focalisées. Alors ne forcez pas les gens à travailler en heures supplémentaires, sauf dans le cas exceptionnel où cela est *vraiment* nécessaire pendant une courte période de temps. En plus, épuiser les gens est Mal.

14

Comment nous faisons les tests

C'est la partie la plus difficile. Je ne sais pas si c'est la partie la plus difficile de Scrum, ou juste la partie la plus difficile du développement logiciel en général.

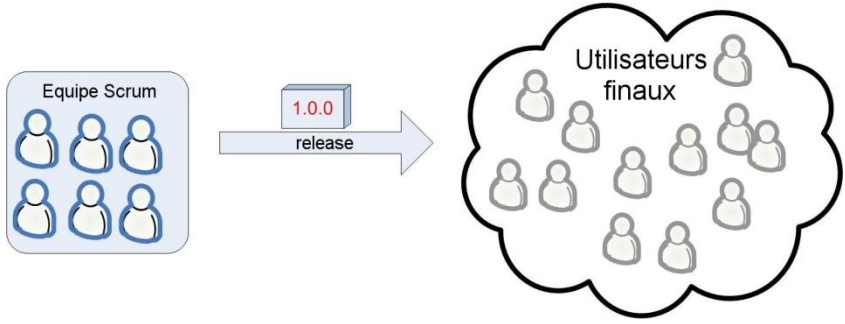
Le test est la partie qui va probablement varier le plus entre différentes organisations. Cela dépendra du nombre de testeurs que vous avez, de l'ampleur de l'automatisation de vos tests, de quel type de système vous avez (juste server+application web ? ou bien vous êtes un éditeur de logiciel vendu dans des boîtes ?), de la durée des cycles de release, de la criticité du logiciel (un serveur de blog vs. un système de contrôle de vol), etc.

Nous avons pas mal expérimenté sur la manière de faire des tests dans Scrum. Je vais essayer de décrire ce que nous avons fait et appris jusque là.

Vous ne pouvez probablement pas éliminer la phase de tests d'acceptation

Dans le monde Scrum idéal, un sprint produit une version de votre système potentiellement déployable. Alors il n'y a qu'à le déployer, n'est-ce-pas ?

Exactement !



Faux.

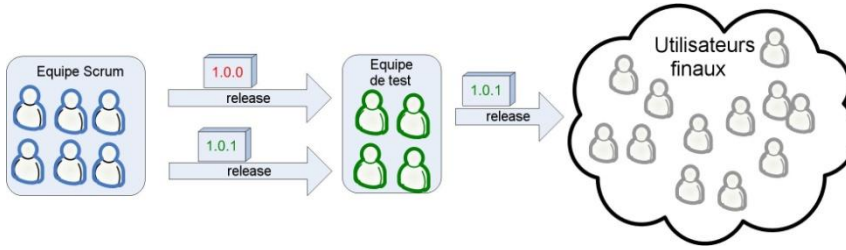
Hein ?

D'après notre expérience, cela ne fonctionne généralement pas. Il y aura de méchants bugs. Si la qualité a une valeur quelconque pour vous, il faut une sorte de phase de test d'acceptation manuelle.

Quel tas de conneries ! Je ne peux pas croire que j'ai écrit ça ! Et puis le livre est devenu viral et les gens du monde entier l'ont lu et ont cru en mes mots. Honte à moi ! Viiiilain auteur, ne fais plus jamais ça ! *claque*

Oui, le test manuel est important et dans une certaine mesure inévitable. Mais il devrait être effectué *par l'équipe dans le sprint*, pas refilé à un groupe séparé ou préservé pour une future phase de test. C'est pourquoi j'ai balancé le modèle en cascade, vous vous rappelez ?

C'est le moment où des testeurs dédiés, qui ne font *pas* partie de l'équipe, martèlent le système avec ces types de tests que l'équipe Scrum n'avait pas imaginés, ou n'avait pas eu le temps de faire, ou pour lesquels elle n'avait pas le matériel nécessaire. Les testeurs accèdent au système exactement comme les utilisateurs finaux, ce qui signifie qu'ils doivent le faire manuellement (en supposant que votre système est destiné à des utilisateurs humains).



L'équipe de test va trouver des bugs, l'équipe Scrum va devoir faire des releases de corrections de bugs, et tôt ou tard (j'espère tôt) vous pourrez livrer aux utilisateurs finaux une version corrigée 1.0.1, au lieu de la version instable 1.0.0.

Quand je dis « phase de tests d'acceptation », je fais référence à la période complète de tests, débogage, et nouvelles releases jusqu'à ce qu'il y ait une version suffisamment bonne pour la production.

Minimisez la phase de tests d'acceptation

La phase de tests d'acceptation fait mal. On sent clairement qu'elle n'est pas agile.

Exactement ! Alors ne la faites pas.

Bon, OK, je sais. Dans certains environnements, cela peut paraître inévitable. Mais mon point est que j'avais l'habitude de penser que c'était inévitable. Mais maintenant, j'ai vu comment des sociétés très agiles avançaient vite *et* augmentaient leur niveau de qualité en se débarrassant de la phase séparée de tests d'acceptation et en fusionnant ce travail dans le sprint. Alors si vous pensez que c'est inévitable, cela peut être que vous êtes aveuglé par votre status quo (comme je l'étais). Néanmoins, ce chapitre fournit des modèles utiles sur comment faire avec des tests d'acceptation séparés, comme mesure temporaire jusqu'à ce que vous réussissiez à fusionner le tout dans un sprint. :o)

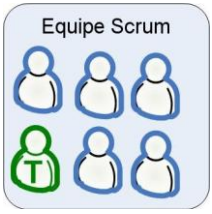
Bien qu'il ne soit pas possible de l'éliminer, nous pouvons (et nous y parvenons) essayer de la minimiser. Plus précisément, de minimiser la quantité de *temps* nécessaire pour la phase de tests d'acceptation. Ce résultat est obtenu en :

- Maximisant la qualité du code produit par l'équipe Scrum
- Maximisant l'efficacité du travail de test manuel (c'est-à-dire trouver les meilleurs testeurs, leur donner les meilleurs outils, s'assurer qu'ils rapportent les tâches consommatrices de temps qui pourraient être automatisées)

Alors comment maximisons-nous la qualité du code produit par l'équipe Scrum ? Eh bien, il y a plein de manières. En voici deux qui marchent très bien d'après nous :

- Mettre des testeurs dans l'équipe Scrum
- En faire moins par sprint

Augmenter la qualité en mettant des testeurs dans l'équipe Scrum



Oui, j'entends bien les deux objections :

- « Mais c'est évident ! Les équipes Scrum sont supposées être *multifonctions*. »
- « Les équipes Scrum sont supposées être sans rôle. On ne peut pas avoir quelqu'un qui serait *uniquement* un testeur ! »

Laissez-moi clarifier. Ce que je veux dire par « testeur » dans ce cas est « une personne dont le talent principal est le test », plutôt que « une personne dont le rôle est seulement de tester ».

C'est un point important. "Multifonction" ne veut pas dire que tout le monde sait tout faire. Cela veut juste dire que tout le monde est enclin à faire plus que juste son propre truc. On veut un joli mix de spécialisation et de multifonction. Alors toute l'équipe est collectivement responsable pour la qualité de son produit, et pratiquement tout le monde sera impliqué dans le test. Le testeur, cependant, guidera le travail, travaillera en paire avec les développeurs sur l'automatisation des tests, et fera personnellement les tests manuels les plus complexes.

Les développeurs sont souvent d'assez mauvais testeurs. *Surtout* les développeurs testant leur propre code.

Le testeur est le « gars qui signe officiellement »

En plus d'être « juste » un membre de l'équipe, le testeur a un boulot important. C'est le gars qui décide quand c'est terminé. Rien n'est considéré comme « terminé » dans un sprint tant que *lui* n'a pas dit que c'était terminé. J'ai découvert que les développeurs disent souvent que quelque chose est terminé alors que ce ne l'est pas réellement. Même si vous avez une définition de « terminé » très claire (et vous devriez vraiment en avoir une, voir page 40 « la définition de 'terminé' »), les développeurs vont l'oublier fréquemment. Nous programmeurs sommes des gens impatients et nous désirons passer à la tâche suivante dès que possible.

Mais alors comment M. T (notre testeur) sait-il que quelque chose est terminé ? Eh bien, avant tout, il devrait (surprise) le *tester* ! Dans beaucoup de cas, il s'avère que quelque chose considéré comme « terminé » par un développeur n'est même pas *testable*. Parce que son travail n'a pas été enregistré dans le système de gestion de sources, parce qu'il n'a pas été déployé sur le serveur de tests, ou parce qu'il n'est pas possible de l'exécuter, etc. Une fois que M. T a testé la fonctionnalité, il devrait parcourir la check-list définissant « terminé » (si vous en avez une) avec le développeur. Par exemple si la définition de « terminé » exige qu'il devrait y avoir une note de release, alors M. T vérifie qu'il y a bien une note de release. S'il existe une sorte de spécification plus formelle pour la fonctionnalité (c'est rare dans notre cas) alors M. T vérifie cela également. Etc.

Un effet de bord positif de tout cela est que l'équipe a maintenant un gars qui est parfaitement au point pour organiser la démonstration du sprint.

Je ne suis plus un grand fan du modèle du gars qui signe officiellement. Cela introduit un goulot d'étranglement et met trop de responsabilité dans les mains d'une personne. Mais je peux voir son utilité dans certaines circonstances (c'était sûrement utile à ce moment là). Aussi, si quiconque devait signer officiellement sur la qualité, ce devrait être le véritable utilisateur.

Que fait le testeur quand il n'y a rien à tester ?

Cette question est posée sans arrêt. M. T : « Hé Scrum master, il n'y a rien à tester en ce moment, alors qu'est-ce que je dois faire ? » Il peut falloir une semaine avant que l'équipe ne termine la première histoire, donc que devrait faire le testeur pendant *ce* temps ?

Eh bien, tout d'abord, il devrait *préparer les tests*. C'est-à-dire écrire les spécifications de test, préparer un environnement de test, etc. Donc quand un développeur a quelque chose de prêt à tester, il ne devrait pas y avoir d'attente, M. T devrait commencer à tester immédiatement.

Si l'équipe pratique le DDT alors des gens passent du temps à écrire du code de test depuis le premier jour. Le testeur devrait programmer en duo avec les développeurs qui écrivent du code de test. Si le testeur ne sait pas programmer, alors il devrait tout de même programmer en duo avec les développeurs, sauf qu'il devrait seulement jouer le rôle du navigateur et laisser le clavier au développeur. Un bon testeur imagine généralement des types de test différents de ceux d'un bon développeur, si bien qu'ils sont complémentaires l'un de l'autre.

Si l'équipe ne pratique pas le DDT, ou s'il n'y a pas assez d'écriture de cas de tests pour occuper tout le temps du testeur, il devrait simplement faire ce qu'il peut pour aider l'équipe à atteindre le but du sprint. Tout comme n'importe quel autre membre de l'équipe. Si le testeur peut programmer c'est super. Sinon, votre équipe devra identifier toutes les tâches hors programmation qui doivent être faites dans le sprint.

Durant la décomposition des histoires en tâches pendant la réunion de planning de sprint, l'équipe a tendance à se focaliser sur les *tâches de programmation*. Toutefois, habituellement il y a beaucoup de *tâches hors programmation* qui doivent être faites dans le sprint. Si vous passez du temps à essayer d'*identifier les tâches hors programmation* durant la phase de planning du sprint, il y a des chances pour que M. T soit capable de contribuer pas mal, même s'il ne sait pas programmer et s'il n'y a pas de tests à faire tout de suite.

Exemples de tâches hors programmation qui doivent souvent être faites dans un sprint :

- Mettre en place un environnement de test.

- Eclaircir les spécifications.
- Discuter les détails de déploiement avec le service Opérations.
- Ecrire les documents de déploiement (notes de release, demandes de commentaires, ou quoi que ce soit que votre organisation fasse).
- Gérer les contacts avec des ressources externes (concepteurs d'IHM par exemple).
- Améliorer les scripts de build.
- Raffiner la décomposition des histoires en tâches.
- Identifier les questions clés posées par les développeurs et obtenir des réponses.

D'un autre côté, que faisons-nous si M. T devient un goulet d'étranglement ? Disons que nous sommes au dernier jour du sprint et que soudainement beaucoup de choses sont faites et que M. T n'a aucune chance de tout tester. Que faisons-nous ? Eh bien nous pourrions transformer tout le monde dans l'équipe en assistants de M. T. Il décide ce qu'il doit faire lui-même, et il délègue le travail de base au reste de l'équipe. Voilà en quoi consiste une équipe multifonctionnelle !

Donc oui, M. T *joue* un rôle spécial dans l'équipe, mais il est tout de même autorisé à faire d'autres travaux, et les autres membres de l'équipe sont tout de même autorisés à faire son travail.

Bien dit ! (Je me permets de m'auto-congratuler des fois, OK ?) Et c'est aussi une bonne façon d'avoir une vision sur toutes les autres compétences de l'équipe.

Augmenter la qualité en en faisant moins par sprint

On en revient à la réunion de planning de sprint. Pour faire court, n'essayez pas de faire rentrer trop d'histoires dans le sprint. Si vous avez des problèmes de qualité, ou de longs cycles de test d'acceptation, faites en moins par sprint ! Cela va presque automatiquement conduire à une qualité plus élevée, des cycles de test d'acceptation plus courts, moins de bugs affectant les utilisateurs finaux, et une productivité plus élevée dans le long terme puisque l'équipe peut se focaliser tout le temps sur de nouveaux trucs au lieu de corriger de vieux trucs qui cassent sans arrêt.

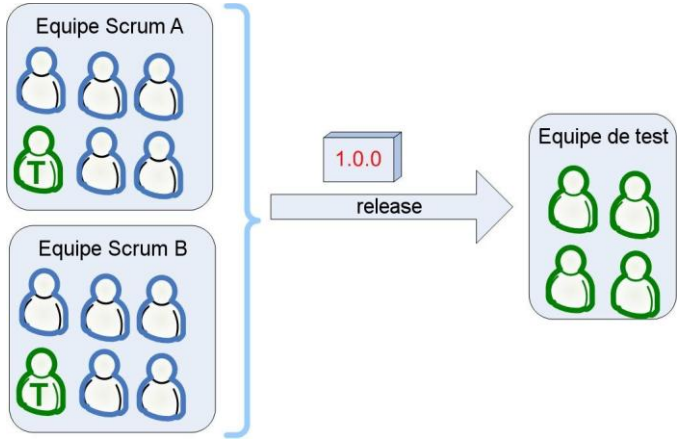
C'est presque toujours plus rentable de construire moins, mais de le construire stable, plutôt que de construire beaucoup de choses et d'ensuite avoir à faire des hot-fix dans la panique.

C'est juste totalement vrai ! Je continue à le voir encore et encore dans les équipes du monde entier.

Est-ce que les tests d'acceptation devraient faire partie du sprint ?

Nous hésitons beaucoup ici. Certaines de nos équipes incluent les tests d'acceptation dans le sprint. Toutefois la plupart de nos équipes ne le font pas, pour deux raisons :

- Un sprint est à durée limitée. Le test d'acceptation (selon ma définition qui inclut le débogage et les nouvelles releases) est très difficile à limiter dans le temps. Que faire si le délai est dépassé et que vous avez toujours un bug critique ? Allez-vous livrer le produit en production avec un bug critique ? Allez-vous attendre jusqu'au sprint suivant ? Dans la plupart des cas les deux solutions sont inacceptables. Donc nous n'y incluons pas le test d'acceptation manuel.
- Si vous avez plusieurs équipes Scrum travaillant sur le même produit, le test d'acceptation manuel doit être effectué sur le résultat combiné des deux équipes. Si les deux équipes ont fait l'acceptation manuelle durant le sprint, vous avez quand même besoin d'une équipe pour tester la release finale, qui est le build intégrant le travail des deux équipes.

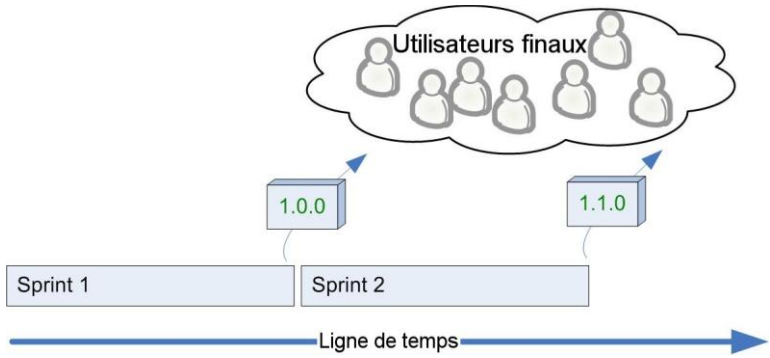


Cela n'est en aucune façon une solution parfaite mais elle est suffisamment bonne pour nous dans la plupart des cas.

Encore une fois, efforcez-vous à faire des tests d'acceptation dans chaque sprint. Cela prend du temps pour y arriver, mais vous ne le regretterez pas. Même si vous n'y arrivez jamais, le fait d'essayer vous permettra d'effectuer de nombreux progrès dans la manière dont vous travaillez.

Des cycles de sprints vs. des cycles de test d'acceptation

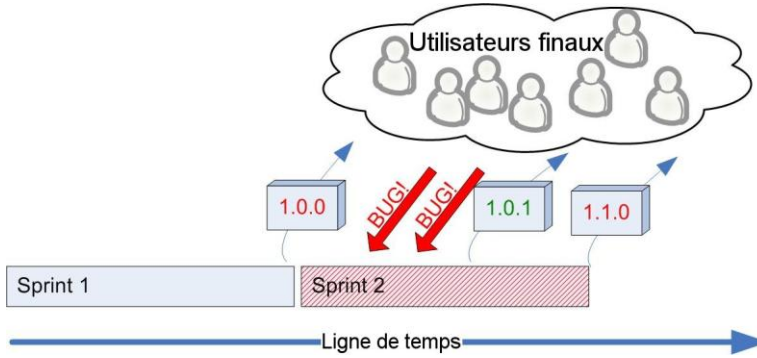
Dans un monde McScrum parfait vous n'avez pas besoin de phases de test d'acceptation puisque chaque équipe Scrum livre une nouvelle version de votre système prête pour la production après chaque sprint.



C'est possible ! J'ai vu de réelles équipes livrer en production tous les jours, parfois même plusieurs fois par jour. Lorsqu'un formateur Scrum leur dit « Vous devez avoir un incrément de produit potentiellement livrable et entièrement testé à chaque fin de sprint », leur réaction est « Hein ? Pourquoi attendre aussi longtemps ? »

Alors non, vous n'avez pas besoin d'un monde McScrum parfait pour cela. Remontez juste vos manches, trouvez ce qui vous empêche d'avoir du code livrable à chaque sprint, et résolvez les problèmes un par un. Bien sûr, cela peut être plus ou moins difficile selon votre domaine, mais cela vaut tout de même la peine d'être essayé. Prenez juste votre cycle de release comme il est aujourd'hui (qu'il soit mensuel, annuel ou quoi que ce soit), et raccourcissez le graduellement, mais en continu.

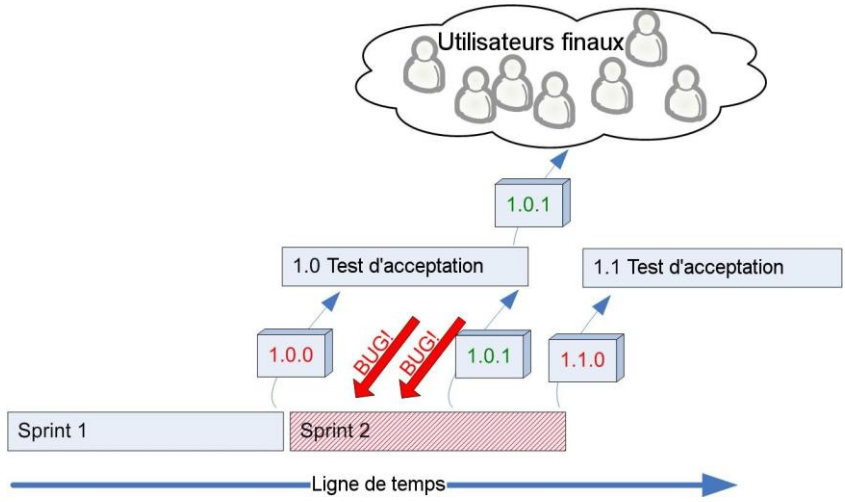
Eh bien voici une image plus réaliste :



Après le sprint 1, une version 1.0.0 boguée est livrée. Durant le sprint 2, des rapports de bugs commencent à arriver et l'équipe passe la majeure partie de son temps à débugger et est forcée de faire à mi-sprint une release 1.0.1 de correction de bugs. Ensuite à la fin du sprint 2 ils livrent une nouvelle version 1.1.0 avec de nouvelles fonctionnalités, qui est bien sûr encore plus boguée puisqu'ils ont eu encore moins de temps pour la faire correctement cette fois-ci étant donné toutes les perturbations venant de la release précédente. Etc etc.

Les hachures rouges dans le sprint 2 symbolisent le chaos.

Pas très joli n'est-ce pas ? Eh bien, ce qui est triste c'est que le problème perdure même si vous avez une équipe de test d'acceptation. La seule différence est que la plupart des rapports de bugs vont venir de l'équipe de test au lieu de clients finaux mécontents. C'est une énorme différence sur le plan commercial, mais pour les développeurs cela revient à peu près à la même chose. Sauf que les testeurs sont habituellement moins agressifs que les utilisateurs finaux. Habituellement.



Nous n'avons trouvé aucune solution simple à ce problème. Cependant nous avons beaucoup expérimenté avec différents modèles.

Avant tout, à nouveau, il faut maximiser la qualité du code que l'équipe Scrum délivre. Le coût de trouver et corriger les bugs très tôt, durant un sprint, est bien plus faible en comparaison du coût de les trouver et corriger après.

Mais il reste le fait que, même si on peut minimiser le nombre de bugs, il y aura toujours des rapports de bugs qui arriveront après qu'un sprint soit terminé. Comment gérons-nous cela ?

Approche 1 : « Ne commencez pas à construire de nouvelles choses avant que les anciennes ne soient en production »

Ca a l'air bien n'est-ce pas ? Avez-vous ressenti vous aussi ce sentiment de satisfaction ?

Oui, c'est super !

Nous avons été proches d'adopter cette approche plusieurs fois, et avons conçu de beaux modèles sur la manière de le faire. Toutefois nous avons toujours changé d'avis quand nous avons réalisé l'inconvénient. Il nous faudrait ajouter une période de release sans limite de temps entre les sprints, pendant laquelle nous ferions seulement du test et du débogage jusqu'à ce que nous puissions faire une release de production.

Pas si votre définition de terminé est "en production". Dans ce cas, vous pouvez commencer le prochain sprint immédiatement, vu que le code du dernier sprint est déjà en production. Il a été livré en continu pendant le sprint. Un peu extrême, oui, mais c'est faisable.

Nous n'avons pas aimé l'idée d'avoir des périodes sans limite de temps entre les sprints, principalement parce que cela casserait le rythme régulier des sprints. Il ne serait plus possible de dire « toutes les 3 semaines nous démarrons un nouveau sprint ». De plus, cela ne résout pas complètement le problème. Même si nous avons une période de release, il y aura toujours des rapports de bugs urgents qui arriveront de temps en temps, et il nous faut être prêt à les gérer.

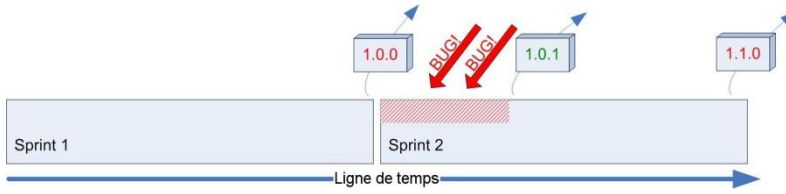
C'est en effet vrai. Même si vous vous débrouillez pour livrer en continu, vous avez tout de même besoin d'un moyen pour traiter les anomalies urgentes qui surviendront. Parce que cela arrivera parfois. Aucune équipe n'est *si* bonne qu'elle n'en aura pas. Et la meilleure façon de traiter cela est de laisser un peu de mou dans votre sprint.

Approche 2 : « OK pour commencer à construire de nouvelles choses, mais priorisez la mise en production des anciennes »

C'est notre approche préférée. En tout cas pour le moment.

Fondamentalement, quand nous finissons un sprint nous commençons le suivant. Mais nous nous attendons à passer un certain temps dans le sprint suivant à corriger des bugs du dernier sprint. Si le sprint suivant est sérieusement endommagé à cause du temps qu'il a fallu pour corriger des bugs du sprint précédent, nous évaluons pourquoi cela est arrivé et comment nous pouvons améliorer la qualité. Nous nous assurons que les sprints sont suffisamment longs pour survivre à une bonne période de correction de bugs du sprint précédent.

Progressivement, sur une période de pas mal de mois, la quantité de temps passée à corriger des bugs des sprints précédents a diminué. De plus nous avons pu impliquer moins de gens quand des bugs arrivaient, si bien qu'il n'était pas nécessaire de perturber *l'ensemble* de l'équipe à chaque fois. Maintenant nous sommes à un niveau plus acceptable.



Durant les réunions de planning de sprint nous fixons le facteur de focalisation à une valeur suffisamment basse pour prendre en compte le temps que nous nous attendons à passer à corriger des bugs du sprint dernier. Avec le temps les équipes sont devenues assez bonnes à cette estimation. La métrique de vélocité aide beaucoup (voir page 31 « Comment l'équipe décide quelles histoires inclure dans le sprint ? »).

Ou utilisez juste la météo de la veille – tirez seulement autant de points d'histoire que vous avez complétés au dernier sprint ou la moyenne des points complétés sur les trois derniers sprints. Ainsi votre sprint va automatiquement avoir un mou intégré pour gérer ces perturbations et ces corrections à chaud. Vous allez automatiquement limiter le travail à la capacité, et allez plus vite en conséquence (lisez n'importe quel livre sur le lean ou la théorie des files d'attente si vous avez besoin d'être convaincu que de surcharger un sprint est une mauvaise idée).

Mauvaise approche – « se focaliser sur la construction de nouvelles choses »

Cela signifie en effet «se focaliser sur la construction de nouvelles choses *plutôt que d'arriver à mettre les anciennes en production*». Qui voudrait faire ça ? Pourtant nous avons souvent fait cette erreur au début, et je suis sûr que beaucoup d'autres entreprises l'ont faite aussi. C'est une maladie liée au stress. Beaucoup de managers ne comprennent pas que, quand tout le codage est fini, vous êtes généralement encore loin de la release de production. Du moins pour les systèmes complexes. Donc le manager (ou le directeur de produit) demande à l'équipe de continuer à ajouter de nouvelles choses alors que le fardeau de vieux code presque-prêt-pour-la-release devient de plus en plus lourd, et ralentit tout.

Je suis constamment ébahi par le nombre de sociétés qui tombent dans ce piège. Tout ce qu'elles ont à faire est de limiter le nombre de projets ou de fonctionnalités en cours. J'ai vu des cas où des sociétés devenaient littéralement sept fois plus rapides en faisant cela. Imaginez cela – tout autant de trucs produits, mais sept fois plus vite, sans avoir à travailler plus dûr ni embaucher des gens. Et une meilleure qualité également, grâce à des boucles de feedback plus courtes. Fou mais vrai.

Ne distancez pas le maillon le plus lent de votre chaîne

Disons que le test d'acceptation est votre maillon le plus lent. Vous n'avez pas assez de testeurs, ou la période de test d'acceptation dure longtemps à cause la qualité lamentable du code.

Disons que votre équipe de test d'acceptation peut tester au plus 3 fonctionnalités par semaine (non, nous n'utilisons pas les « fonctionnalités par semaine » comme métrique ; j'utilise le terme juste pour cet exemple). Et disons que vos développeurs peuvent développer 6 nouvelles fonctionnalités par semaine.

Il va être tentant pour les managers ou directeurs de produit (ou peut être même pour l'équipe) de planifier le développement de 6 nouvelles fonctionnalités par semaine.

Ne le faites surtout pas ! La réalité va vous rattraper d'une manière ou d'une autre, et ca va faire mal.

Plutôt, planifiez 3 nouvelles fonctionnalités par semaine, et passez le reste du temps à réduire le goulet d'étranglement du test. Par exemple :

- Faire travailler quelques développeurs comme testeurs (oh ils vont vous adorer pour cela…).
- Implémenter des outils et scripts qui rendent le test plus facile.
- Ajouter plus de code de test automatisé.
- Augmenter la longueur des sprints et inclure le test d'acceptation dans les sprints.
- Définir certains sprints comme des «sprints de test» pendant lesquels l'équipe complète travaille comme une équipe de test d'acceptation.
- Embaucher plus de testeurs (même si cela signifie supprimer des postes de développeurs)

Nous avons essayé toutes ces solutions (sauf la dernière). La meilleure solution à long terme est bien sûr les points 2 et 3, c'est-à-dire de meilleurs outils et scripts ainsi que l'automatisation des tests.

Les rétrospectives sont un bon forum pour identifier le maillon le plus lent dans la chaîne.

Cela devient auto-ajusté si les tests d'acceptation sont inclus dans le sprint, plutôt que faits séparément. Essayez-le – incluez vos tests d'acceptation dans votre définition de terminé, et voyez ce qu'il se passe avec le temps.

Retour à la réalité

Je vous ai probablement donné l'impression que nous avons des testeurs dans toutes les équipes Scrum, que nous avons de grosses équipes de test d'acceptation pour chaque produit que nous livrons après chaque sprint, etc.

Eh bien, nous ne les avons pas.

Nous avons *quelquefois* réussi à faire tout cela, et nous en avons vu les effets positifs. Mais nous sommes encore loin d'un processus d'assurance qualité acceptable, et nous avons encore beaucoup à apprendre dans ce domaine.

En effet, nous avions beaucoup de choses à apprendre. :o)

15

Comment nous gérons les équipes multi-Scrum

Beaucoup de choses deviennent plus difficiles quand vous avez plusieurs équipes Scrum qui travaillent sur le même projet. Ce problème est universel et n'a pas vraiment de lien avec Scrum. Plus de développeurs = plus de complications.

J'ai beaucoup travaillé avec la mise à l'échelle depuis que j'ai écrit ce livre. Pour un exemple détaillé, regardez *Lean depuis les tranchées.* Ce livre a un style très similaire à celui-ci. Il illustre comment un projet gouvernemental de 60 personnes a été effectué en utilisant une combinaison de Scrum, Kanban et XP. https://pragprog.com/book/hklean/lean-from-the-trenches

Nous avons (comme d'habitude) fait des essais. Au plus nous avions une équipe de 40 personnes environ qui travaillaient sur le même projet.

Les questions clés sont :

- Combien d'équipes faut-il créer ?
- Comment répartir les personnes dans les équipes ?

Et, bien sûr, comment garder les équipes synchronisées entre-elles.

Combien d'équipes faut-il créer

Si traiter avec plusieurs équipes Scrum est si dur, pourquoi s'embêter ? Pourquoi ne pas mettre tout le monde dans la même équipe ?

La plus grosse équipe Scrum que nous avons eu faisait environ 11 personnes. Ca fonctionnait, mais pas très bien. Les mêlées quotidiennes avaient tendance à trainer au-delà de 15 minutes. Les membres de l'équipe ne savaient pas ce que les autres faisaient, il pouvait y avoir de la confusion. C'était difficile pour le Scrum master de garder tout le monde concentré sur l'objectif, et de trouver le temps d'adresser tous les obstacles qui étaient remontés.

L'alternative est de diviser l'équipe en deux. Mais est-ce mieux ? Pas nécessairement.

Si l'équipe est expérimentée et confortable avec Scrum, et s'il y a une approche logique de découper la feuille de route en deux pistes distinctes, et que les deux pistes ne concernent pas le même code source, alors je dirais que c'est une bonne idée de diviser l'équipe. Sinon je considérerais de rester avec une équipe, malgré le désavantage des grosses équipes.

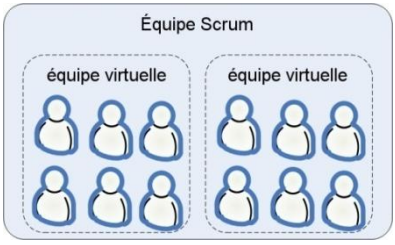
Mon expérience est qu'il est préférable d'avoir peu d'équipes même trop grosses que d'avoir plein de petites équipes qui interfèrent les unes les autres. Créez de petites équipes seulement si elles n'ont pas besoin d'interagir entre elles !

Equipes virtuelles

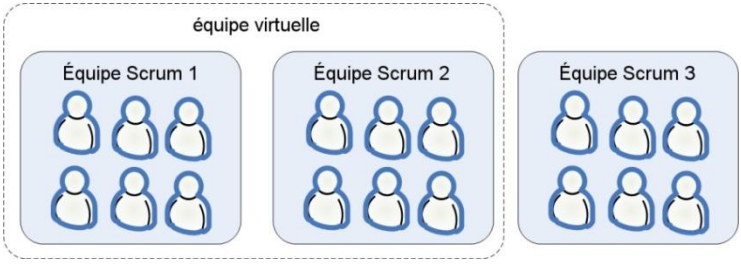
Comment savoir si vous prenez la bonne ou mauvaise décision par rapport au compromis entre « grosses équipes » vs « beaucoup d'équipes » ?

Si vous gardez les yeux et les oreilles ouvertes vous aurez remarqué la formule « équipes virtuelles ».

Exemple 1: Vous choisissez d'avoir une grande équipe. Mais quand vous regardez qui parle à qui pendant l'itération, vous notez que l'équipe est effectivement divisée en deux sous-équipes.



Exemple 2*:* Vous choisissez d'avoir trois équipes plus petites. Mais quand vous regardez qui parle à qui pendant l'itération, vous notez que les équipes 1 et 2 discutent entre elles tout le temps, tandis que l'équipe 3 travaille de manière isolée.



Qu'est-ce que ça signifie ? Que votre stratégie de division des équipes n'est pas bonne ? Oui, si les équipes virtuelles semblent plutôt permanentes. Non, si vos équipes virtuelles semblent être temporaires.

Regardez l'exemple 1. Si les deux sous-équipes virtuelles ont tendance à changer de temps en temps (c'est-à-dire que les personnes bougent entre les sous-équipes virtuelles) alors vous avez probablement pris la bonne décision en les regroupant dans une équipe Scrum unique. Si les deux sous-équipes restent les mêmes pendant toute l'itération, vous voudrez probablement les séparer en deux vraies équipes Scrum pour la prochaine itération.

Maintenant regardez l'exemple 2. Si l'équipe 1 et 2 discutent entre elles (mais pas l'équipe 3) durant toute l'itération, vous voudrez probablement regrouper l'équipe 1 et l'équipe 2 en une équipe Scrum unique à la prochaine itération. Si l'équipe 1 et l'équipe 2 discutent beaucoup entre elles pendant la première moitié de l'itération, et qu'ensuite l'équipe 1 et l'équipe 3 discutent entre elles pendant la seconde moitié de l'itération, alors vous devriez considérer de fusionner les trois équipes en une, ou juste les laisser en trois équipes. Abordez la question durant la rétrospective d'itération et laissez les équipes décider elles-mêmes.

La division des équipes est l'une des parties vraiment difficiles de Scrum. Ne réfléchissez pas trop ou n'optimisez pas trop. Expérimentez, gardez un œil sur les équipes virtuelles, et soyez sûr que vous prenez suffisamment de temps pour discuter de ce genre de choses pendant les rétrospectives. Tôt ou tard vous trouverez la bonne solution à votre situation. Le plus important est que les équipes soient confortables et n'hésitez pas entre les deux trop souvent.

Une technique avancée de plus en plus populaire est de laisser les équipes se reformer dynamiquement au besoin. J'appelle parfois cela le modèle « super équipe » ou « sous-équipes dynamiques ». Cela fonctionne mieux lorsque vous avez 12-16 personnes installées à côté, qui se connaissent bien, et qui travaillent sur le même produit. Donnez-leur un unique backlog de produit partagé, et demandez-leur de former des petits groupes autour d'histoires individuelles (« Pat, Tom et moi allons travailler sur l'histoire X maintenant »), et de se regrouper dynamiquement au fur et à mesure que les histoires sont terminées. Un peu de facilitation est nécessaire pour que cela marche en pratique, mais c'est parfait dans le cas où vous ne voulez pas d'une seule grosse équipe, et que vous n'avez pas encore trouvé de moyen consistant de la découper en plusieurs petites équipes.

La taille optimale de l'équipe

La plupart des livres que j'ai lus affirment que la taille « optimale » de l'équipe est quelque part autour de 5 – 9 personnes.

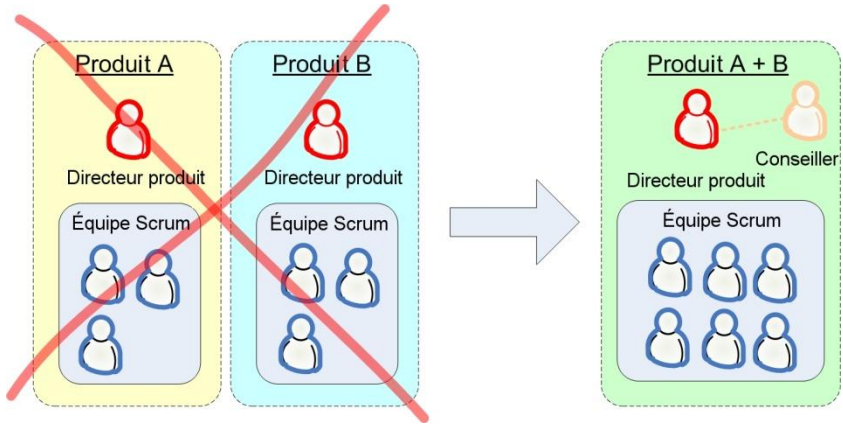
D'après ce que j'ai vu jusqu'ici je ne peux qu'être d'accord. Même si je dirai plutôt 3 – 8 personnes. En fait, je crois que ça vaut le coût d'atteindre des équipes de cette taille.

Disons que vous avez une unique équipe Scrum de 10 personnes. Considérons que l'on éjecte les deux membres les plus médiocres. Oups, ai-je vraiment dit ça ?

MDR. Comment est-ce que j'ai pu m'en tirer avec ça ? :o)

Cependant, j'ai remarqué que parfois une grosse équipe bouge plus vite lorsque certains membres sont en vacances. Mais ce n'est généralement pas parce que les personnes absentes étaient des maillons faibles. C'est juste que la taille d'équipe devint plus gérable, et que la communication et la confusion supplémentaires étaient réduites.

Disons que vous avez deux produits différents, avec une équipe de 3 personnes par produit, et les deux avancent trop lentement. Ca *peut* être une bonne idée de les fusionner en une seule équipe de 6 personnes responsable des deux produits. Dans ce cas laissez partir l'un des deux directeurs de produit (ou donnez lui un rôle consultatif ou autre chose).



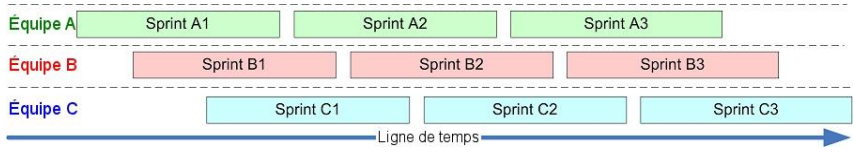
Disons que vous avez une seule équipe Scrum de 12 personnes, car le code est dans un état tellement lamentable qu'il n'y a pas moyen pour 2 équipes différentes de travailler dessus indépendamment. Faites de sérieux efforts pour corriger le code (au lieu de construire de nouvelles fonctionnalités) jusqu'à ce que vous puissiez diviser l'équipe. Cet investissement sera probablement amorti rapidement.

Cela vaut la peine d'être souligné. Si vous avez du mal à trouver la bonne structure d'équipe, le véritable coupable est souvent l'architecture système. Une bonne architecture laisse les équipes évoluer rapidement et indépendamment, une mauvaise architecture fait trébucher les équipes les unes sur les autres et s'enlise de dépendances. Alors vous devriez continuellement questionner et faire évoluer à la fois votre architecture et votre structure d'équipe.

Sprints synchronisés – ou non ?

Disons que vous avez trois équipes Scrum qui travaillent sur le même projet. Leurs sprints devraient-ils être synchronisés, c'est-à-dire commencer ou finir ensemble ? Ou devraient-ils se chevaucher ?

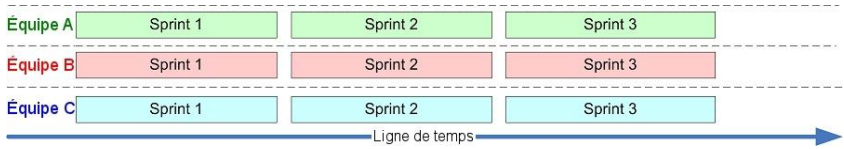
Notre première approche était les sprints qui se chevauchent (en respectant les durées).



Ca sonne bien. A n'importe quel moment il y aurait un sprint sur le point de finir et un nouveau sprint sur le point de commencer. La charge de travail du directeur de produit serait également répartie dans le temps. Il y aurait des nouvelles versions en continu. Des démonstrations toutes les semaines. Alléluia.

Oui, je sais, mais ça *semblait* vraiment convaincant à l'époque !

Nous avions juste commencé à faire ça quand un jour j'ai eu l'opportunité de parler avec Ken Schwaber (en conjonction avec ma certification Scrum). Il fit remarquer que c'était une *mauvaise* idée, qu'il serait tellement mieux de synchroniser les sprints. Je ne me rappelle pas ces arguments exacts, mais après la discussion j'étais convaincu.



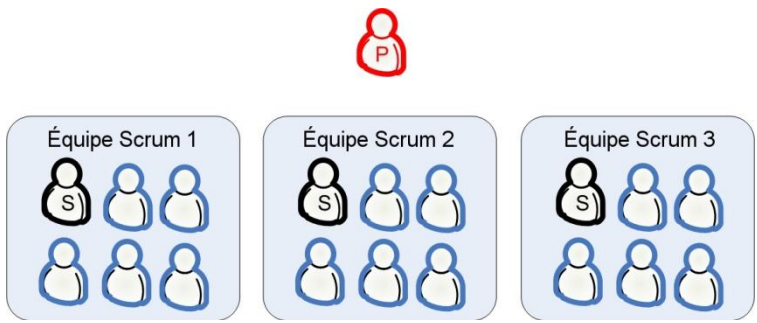
C'est la solution que nous avons utilisée depuis, et nous ne l'avons jamais regretté. Je ne saurai jamais si la stratégie des sprints qui se chevauchent aurait échouée, mais je pense que oui. Les avantages des sprints synchronisés sont :

- Il y a un moment naturel où l'on peut réarranger les équipes – entre les sprints ! En chevauchant les sprints, il n'y a pas moyen de réarranger les équipes sans perturber au moins une équipe au milieu du sprint.
- Toutes les équipes peuvent travailler vers un objectif commun sur un sprint et faire les réunions de planification de sprint ensemble, ce qui mène à une meilleure collaboration entre équipes.
- Il y a moins de travail administratif, c'est-à-dire moins de réunions de planification de sprint, de démonstrations de sprint, et de releases.

Chez Spotify, nous avions l'habitude d'avoir des sprints synchronisés, jusqu'à ce que nous décidions de laisser chaque équipe travailler à son propre rythme et choisir sa propre longueur de sprint (certains font du kanban au lieu de sprints). Lorsque nous avons beaucoup de dépendances entre les équipes, nous faisons des sortes de mêlées-de-mêlées quotidiennes de réunion de synchronisation et une démo hebdomadaire du produit intégré, indépendamment du rythme de chacune des équipes. Je ne peux toutefois pas dire quel modèle est le meilleur– c'est contextuel.

Pourquoi nous avons introduit le rôle du « meneur d'équipe »

Disons que nous avons un seul produit avec trois équipes.



Le gars en rouge marqué P est le Directeur de produit. Les gars en noir marqués S sont les Scrum Masters. Les autres sont les ~~troufions~~... euh... les membres respectables d'équipe.

Avec cette constellation, qui décide quelles personnes devraient être dans quelle équipe ? Le directeur de produit ? Les trois Scrum masters ensemble ? Ou chaque personne choisit son équipe ? Mais alors que faire si tout le monde veut être dans l'équipe 1 (parce que le Scrum master 1 est tellement *avenant*)?

Que faire si plus tard il s'avère impossible d'avoir plus de deux équipes en parallèle sur ce code, et qu'il soit nécessaire de les transformer en 2 équipes de 9 personnes au lieu de des équipes de 6 personnes. Cela signifie 2 Scrum masters. Alors lequel des 3 Scrum masters sera relevé de sa fonction ?

Dans beaucoup d'entreprises ce sont des questions assez sensibles.

Il est tentant de laisser le directeur de produit faire les affectations et réaffectations. Mais ce n'est pas vraiment le travail du directeur de produit, n'est-ce pas ? Le directeur de produit est l'expert métier qui dit à l'équipe dans quelle direction elle doit courir. Il ne devrait pas vraiment être impliqué dans les détails. Surtout s'il s'agit d'un « poulet » (si vous avez entendu la métaphore du poulet et du cochon, sinon tapez « chicken and pig » sous google).

Oh, je déteste cette métaphore stupide (cherchez sur Google si vous devez). Pour une raison étrange, elle était plutôt populaire dans le monde Scrum pour traiter le product owner comme une sorte de tiers qui n'était pas engagé dans le travail qui était fait. Très offensant. Néanmoins, je pense toujours que le product owner ne devrait pas être celui qui pilote la structure de l'équipe, vu que le boulot du product owner est déjà assez difficile. Alors qui devrait le faire ? Continuez à lire…

Nous avons résolu ce problème en introduisant le rôle du « meneur d'équipes ». Cela correspond à ce que vous pourriez appeler le « Scrum des Scrum masters » ou « le chef » ou le « Scrum master principal » etc. Il n'a à mener aucune équipe, mais il est responsable des problèmes transverses aux équipes comme qui devrait être le Scrum master pour chaque équipe, combien de personnes devraient être divisées en équipes, etc.

Nous avons eu du mal à trouver un bon nom pour ce rôle. Le « meneur d'équipes » était le moins mauvais nom que nous avons pu trouver.

Cette solution a bien marché pour nous et je peux la recommander (indépendamment du nom que vous choisissez pour ce rôle).

Dans la plupart des sociétés que j'ai vues, le responsable hiérarchique est responsable de la structure d'équipe, et il n'y a pas besoin d'un rôle de pilotage d'équipe (d'ailleurs, « pilotage d'équipe » est un nom plutôt confusant, vu qu'on dirait qu'il n'y a qu'une seule équipe). Cependant, les meilleurs managers trouvent une manière d'aider l'équipe à s'auto-organiser dans une structure adéquate plutôt que d'assigner une structure top-down.

Comment nous affectons les personnes aux équipes

Il y a deux grandes stratégies pour affecter les personnes aux équipes, lorsque vous avez de multiples équipes sur le même produit.

- Laisser une personne désignée faire l'affectation, par exemple le « meneur d'équipes » que j'ai mentionné ci-dessus, le directeur de produit, ou le manager fonctionnel (s'il est assez impliqué pour pouvoir prendre de bonnes décisions).
- Laisser les équipes le faire eux-mêmes.

Nous les avons essayées toutes les trois. Trois ? Oui. Stratégie 1, Stratégie 2 et une combinaison des deux.

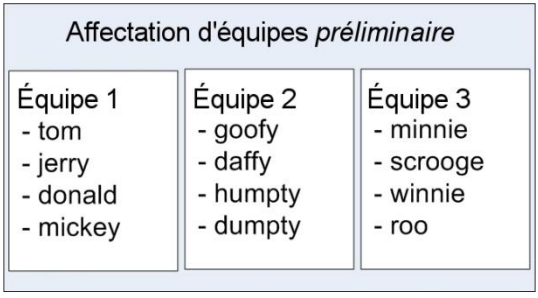
Nous avons trouvé que la combinaison des deux fonctionnait mieux.

Et après davantage d'années d'expérimentation, je ne peux qu'être d'accord.

Avant la réunion de planification de sprint, le meneur d'équipes lance une réunion d'affectation des équipes avec le directeur de produit et tous les Scrum masters. Nous discutons du dernier sprint et décidons si une réaffectation est nécessaire. Peut-être voulons-nous fusionner deux équipes, ou transférer des personnes d'une équipe vers une autre. Nous nous décidons sur quelque chose et nous le mettons par écrit comme *une proposition d'affectation d'équipe*, que nous apportons à la réunion de planification du sprint.

La première chose que nous faisons pendant la réunion de planification de sprint est de parcourir les éléments de plus haute priorité dans le backlog de produit. Le meneur d'équipe dit ensuite quelque chose dans le genre :

« Bonjour tout le monde. Nous suggérons l'affectation d'équipe suivante pour le prochain sprint. »



« Comme vous le voyez, cela signifierait une réduction du nombre d'équipes, passant de 4 à 3. Nous avons listé les membres pour chaque équipe. Groupez-vous s'il-vous-plaît et mettez-vous devant une section du mur. »

(le meneur d'équipes attend pendant que les personnes se baladent dans la pièce, après un moment il y a 3 groupes, chacun se tenant devant une section du mur vide).

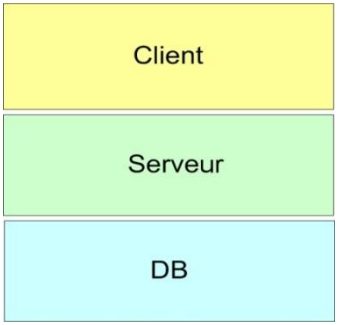
« Maintenant cette division d'équipes est *préliminaire* ! C'est juste un point de départ, pour gagner du temps. Au fur et à mesure que la réunion de planification de sprint progresse vous êtes libre de vous balader vers une autre équipe, de diviser votre équipe en deux, fusionner avec une autre équipe, ou ce que vous voulez. Faites preuve de bon sens en vous basant sur les priorités du directeur de produit. »

C'est ce que nous avons trouvé qui fonctionne le mieux. Un certain niveau de contrôle centralisé au début, suivi par un certain niveau d'optimisations décentralisées après.

C'est un excellent modèle. Gardez juste en tête qu'il y a plein de façons différentes de faire – de le combiner avec le planning de sprint est une façon, et pas toujours la meilleure. De nos jours, je fais généralement les ateliers de réorganisation séparément du planning de sprint, pour garder les réunions de planning de sprint plus focalisées (comme dans, nous avons une structure d'équipe stable et un backlog produit stable quand on planifie le sprint). Pour les grands projets, cependant, il est utile de faire toutes les réunions de planning de sprint simultanément dans une grande salle. Parfois une dépendance complexe peut être résolue simplement en changeant une personne d'équipe (temporairement ou à long terme).

Equipes spécialisées – ou non ?

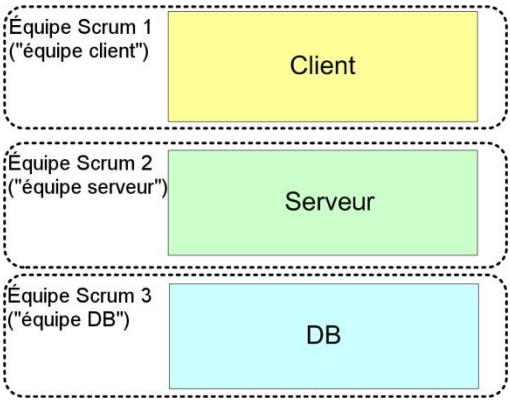
Disons que votre technologie consiste en trois principaux composants :



Et disons que vous avez 15 personnes travaillant sur le produit, aussi vous ne voulez vraiment pas en faire une seule équipe Scrum. Quelles équipes créez-vous ?

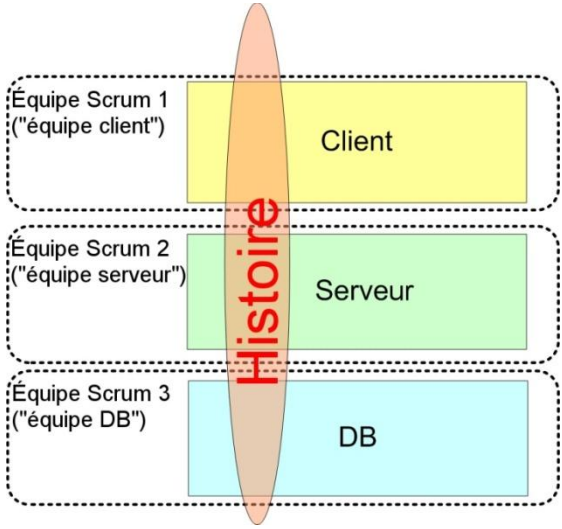
Approche n°1: équipes spécialisées par composant

Une approche est de créer des équipes spécialisées par composant telles qu'une « équipe client », une « équipe serveur » et une « équipe BD ».



C'est comme ça que nous avons commencé. Ca ne fonctionnait pas très bien, du moins pas quand la plupart des histoires impliquaient plusieurs composants.

Par exemple disons que nous avons une histoire appelée « tableau d'affichage où les utilisateurs peuvent poster des messages aux autres ». Cette fonctionnalité de tableau d'affichage impliquerait de mettre à jour l'interface utilisateur côté client, ajouter la logique côté serveur, et ajouter des tables dans la base de données.

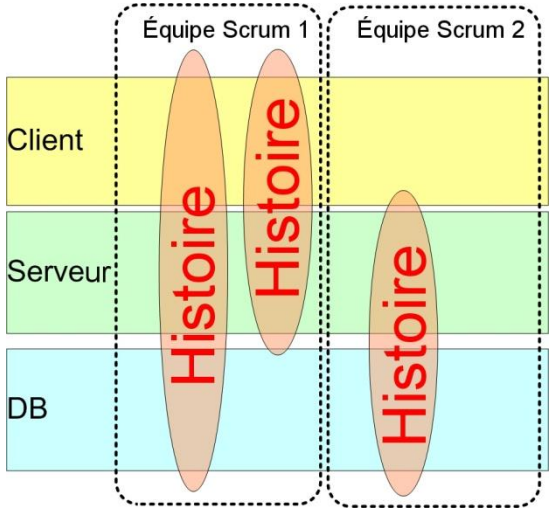


Cela signifie que les trois équipes – l'équipe client, l'équipe serveur, et l'équipe BD – doivent collaborer pour que cette histoire soit terminée. Pas terrible.

Un problème étonnamment commun dans beaucoup de sociétés !

Approche n°2: équipes transversales aux composants

Une seconde approche est de créer des équipes transversales aux composants, c'est-à-dire des équipes qui ne sont liées à aucun composant.



Si beaucoup de vos histoires impliquent plusieurs composants cette stratégie de division d'équipes fonctionnera mieux. Chaque équipe peut implémenter une histoire complète incluant la partie client, la partie serveur et la partie base de données. Les équipes peuvent ainsi travailler plus indépendamment des autres, ce qui est une Bonne Chose.

L'une des premières choses que nous faisons quand nous avons introduit Scrum était de démanteler ces équipes spécialisées par composant (approche 1) et de créer des équipes transversales aux composants à la place (approche 2). Cela réduit le nombre de cas où « nous ne pouvons finir cet élément parce que nous attendons que les gars côté serveur fassent leur partie. »

Même histoire avec presque chaque société avec qui j'ai travaillé. La réorganisation des équipes composant, en équipes fonctionnalité, peut être assez perturbante, mais les bénéfices sont énormes !

Cependant, nous assemblons parfois des équipes temporaires spécialisées par composant quand le besoin est fort.

Parfois, les équipes composant font sens, même pour le long terme. Mais cela devrait être l'exception, pas la norme. Une bonne question test pour une équipe est « qui est votre client, et pouvons-nous satisfaire ses besoins sans avoir à bloquer d'autres équipes ? » Dans la plupart des cas, une équipe fonctionnalité réussira le test et une équipe composant non. L'exception la plus commune sont les équipes s'interfaçant comme les équipes outils et plateforme (ex : une équipe d'infrastructure serveur). Ils devraient traiter l'autre équipe (les équipes fonctionnalité) comme leurs clients dans un sens très littéral. Les grandes sociétés finissent généralement avec un mix d'équipes fonctionnalité orientées vers l'extérieur et des équipes composant orientées vers l'intérieur. Il n'y a toutefois pas de solution miracle, alors continuez à expérimenter !

Réarranger les équipes entre les sprints – ou pas ?

Chaque sprint est généralement assez différent des autres, selon le type des histoires qui ont la plus haute priorité à ce moment particulier. Par conséquent, la création de l'équipe optimale peut être différente pour chaque sprint.

En fait, presque tous les sprints nous disions quelque chose comme « *ce* sprint n'était *pas* vraiment un sprint *normal* parce que (bla bla bla)....». Après quelques temps nous avons abandonné la notion de sprint « normal ». Il n'y a pas de sprint normal. Comme il n'y a pas de famille « normale » ou de personne « normale ».

Sur un sprint ce peut être une bonne idée d'avoir une équipe client seulement, constituée de tous ceux qui connaissent bien le code client. Au prochain sprint ce peut être une bonne idée d'avoir deux équipes transversales en divisant le groupe client.

L'un des aspects clés de Scrum est la « formation de l'équipe », c'est-à-dire si une équipe apprend à travailler ensemble au cours des sprints ils deviendront généralement *très proches*. Ils apprendront à obtenir un *flux de groupe* et atteindront un niveau de productivité incroyable. Mais cela prend quelques sprints pour y parvenir. Si vous n'arrêtez pas de changer les équipes nous n'obtiendrez jamais une équipe vraiment soudée.

Aussi si vous voulez réarranger les équipes, soyez sûr de mesurer les conséquences. Est-ce un changement à long terme ou à court terme ? Si c'est un changement à court terme considérez de le laisser tomber. Si c'est un changement long terme, allez-y.

Une exception lorsque vous commencez à appliquer Scrum avec une grande équipe pour la première fois. Dans ce cas il vaut probablement mieux expérimenter un peu avec les divisions d'équipes jusqu'à ce que vous trouviez quelque chose de confortable pour tous. Assurez-vous que tout le monde comprenne que c'est OK d'avoir tout faux les quelques premières fois, du moment que vous continuez de vous améliorer.

Principe de base : Après quelques sprints, votre structure d'équipe devrait être plutôt stable. Ainsi, pour toute équipe donnée, ses membres seront inchangés (ni ajout ni retrait de personnes) pour au moins un trimestre. Si l'équipe change plus souvent que ça, elle n'atteindra probablement jamais l'état d'hyper-productivité que Scrum vise. Cependant, les changements qui proviennent de l'intérieur (initiés par les membres de l'équipe) sont généralement moins perturbants que les changements d'équipe imposés par au-dessus. Alors si vous êtes manager, efforcez-vous à résister à la tentation de vous immiscer. Laisser les gens focaliser, encouragez la stabilité de l'équipe, mais permettez également aux personnes de changer d'équipes si jamais elles en voient le besoin. Vous serez probablement surpris des résultats !

Les membres d'équipe à temps partiel

Je ne peux que confirmer ce que les livres Scrum disent – avoir des membres d'une équipe Scrum à temps partiel n'est généralement pas une bonne idée.

…comme dans, ça craint vraiment !

Disons que vous êtes sur le point de prendre Joe en tant que membre à temps partiel dans votre équipe Scrum. Réfléchissez bien avant. Avez-vous vraiment besoin de Joe dans votre équipe ? Etes-vous sûr de ne pas pouvoir prendre Joe à temps plein ? Quels sont ses autres engagements ? Est-ce que quelqu'un d'autre peut prendre le relai sur les autres engagements de Joe et laisser à Joe un rôle plus passif, de support, tout en respectant ses engagements ? Est-ce que Joe peut rejoindre votre équipe à plein temps pour le *prochain* sprint, et dans le même temps transférer ses autres responsabilités à quelqu'un d'autre ?

Parfois il n'y a pas de solution. Vous avez désespérément besoin de Joe parce que c'est le seul DBA de l'immeuble, mais les autres équipes ont autant besoin de lui alors il ne sera jamais affecté à plein temps sur votre équipe, et la société ne peut prendre plus de DBAs. Très bien. C'est un cas valide pour l'avoir à temps partiel (c'est d'ailleurs exactement ce qui s'est passé pour nous). Mais assurez-vous de faire cette évaluation à chaque fois.

En général je préfère avoir une équipe de 3 personnes à plein temps que de 8 à temps partiel.

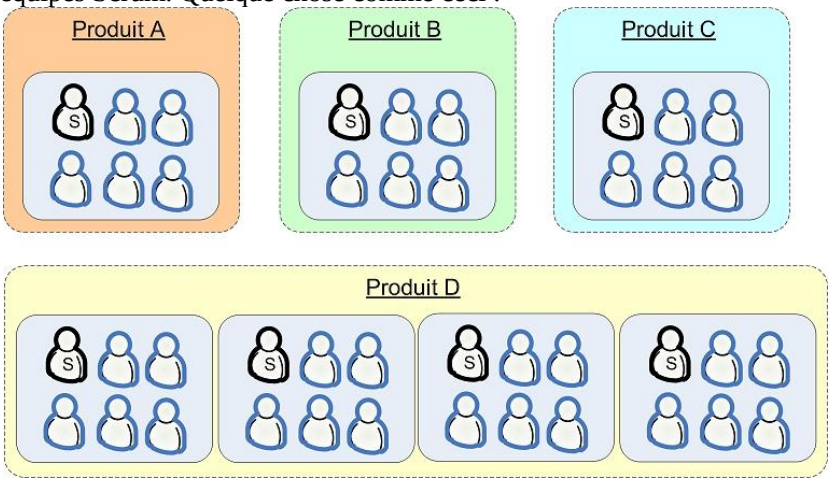
Si vous avez une personne qui va partager son temps entre plusieurs équipes, comme le DBA mentionné plus haut, c'est une bonne idée de l'assigner à l'origine à une équipe. Trouvez l'équipe qui aura le plus besoin de lui, et faites-en son « équipe maison ». Quand personne d'autre ne l'embarque, il assistera aux mêlées quotidiennes de cette équipe, aux réunions de planification de sprint, aux rétrospectives, etc.

Principe de base : Plein temps signifie au moins à 90% dédié à l'équipe. Temps partiel signifie 50-90% (« C'est mon équipe d'origine mais j'ai d'autres engagements aussi »). Moins de 50% signifie que ce n'est pas un membre de l'équipe (« Je peux vous aider les gars de temps en temps, mais je ne fais pas partie de votre équipe et vous ne pouvez pas toujours compter sur moi »). Si une équipe a seulement un temps partiel et le reste est à plein temps, alors ne vous en faites pas. Si elle a plus d'un temps partiel, cependant, vous devriez envisager de réorganiser ou de réduire la quantité totale de travail en cours pour que les temps partiels puissent devenir des temps pleins. Le multitâche est une bête qui tue la productivité, la motivation, et la qualité, alors gardez-le à un minimum absolu !

Comment nous faisons les Scrums-de-Scrums

Le Scrum-des-Scrums est simplement une réunion régulière dans laquelle tous les Scrum masters se rassemblent pour parler.

A un moment donné, nous avions quatre produits, dont trois d'entre eux avait une seule équipe Scrum, et le dernier avait 25 personnes réparties dans plusieurs équipes Scrum. Quelque chose comme ceci :



Cela signifie que nous avions 2 niveaux de Scrums-de-Scrum. Nous avions un Scrum-de-Scrums de « niveau produit » composé des équipes du produit D, et un Scrum-de-Scrums « niveau global » composé de tous les produits.

Scrum-de-Scrums de niveau produit

Cette réunion est très importante. Nous en faisons une par semaine, parfois plus souvent. Nous y discutons des problèmes d'intégration, des problèmes d'équilibrage de la charge des équipes, des préparations pour la prochaine réunion de planification de sprint, etc. Nous y allouons 30 minutes mais elles sont fréquemment dépassées. Une alternative serait d'avoir le Scrum-de-Scrums tous les jours, mais nous n'avons jamais pu réussir à essayer ça.

L'ordre du jour de notre Scrum-de-Scrums est :
1) Tour de table, chacun décrit ce que son équipe a accompli durant la dernière semaine, ce qu'il prévoit de finir cette semaine, et quels sont leurs obstacles.
2) Le moindre souci inter-équipe doit y être rapporté, par exemple des problèmes d'intégration.

L'ordre du jour du Scrum-de-Scrums n'est pas vraiment important pour moi, la chose importante est que vous *teniez* régulièrement les réunions Scrum-de-Scrums.

Chez Spotify, nous le faisons généralement sous forme de "synchronisation quotidienne" entre équipes impliquées dans quelque chose ensemble. Une réunion courte, généralement 15 minutes max. L'ordre du jour est focalisé sur les dépendances plutôt que sur une mise à jour de la situation. Une personne de

chaque équipe parle de ce dont ils ont besoin des autres équipes, s'ils sont bloqués par quoi que ce soit, et ainsi de suite. Nous utilisons parfois un tableau simple avec des post-its pour suivre les dépendances non résolues entre équipes. La réunion devient comme une petite place du marché pour identifier où il y a un besoin de synchronisation. La synchronisation elle-même a lieu séparément – la réunion nous aide juste à trouver qui a besoin de parler à qui pour résoudre les problèmes de dépendances du moment.

Scrum-de-Scrums de niveau global

Nous appelons cette réunion «Le pouls». Nous avons fait cette réunion avec de nombreux format, avec différent types de participants. Tardivement nous avons laissé tomber le concept dans sa globalité et l'avons remplacé par une réunion générale hebdomadaire (eh bien, toutes les personnes impliquées dans le développement) à la place. 15 minutes.

Comment ? 15 minutes ? Réunion générale ? Tous les membres de toutes les équipes de tous les produits ? Est-ce que ça marche ?

Oui ça marche si vous (ou qui que ce soit qui mène la réunion) êtes strict vis-à-vis de sa durée pour la garder courte.

Le format de la réunion est :
1) Les nouvelles récentes du responsable du développement. Des infos sur les évènements à venir par exemple.
2) Tour de table. Une personne de chaque groupe d'un produit indique ce qu'ils ont fini la semaine passée, ce qu'ils comptent finir cette semaine, et le moindre problème. D'autres personnes s'expriment aussi (responsable CM, responsable QA, etc.).
3) N'importe qui d'autre est libre d'ajouter de l'information ou de poser des questions.

Il s'agit d'un forum pour de l'information brève, pas de discussion ni de cogitations. En le laissant comme cela, 15 minutes généralement suffisent. Parfois nous dépassons, mais très rarement plus de 30 minutes au total. Si des discussions intéressantes surviennent je les suspends et invite ceux qui sont intéressés à poursuivre la discussion après la réunion.

Pourquoi faisons-nous une réunion générale « pouls » ? Parce que nous avons remarqué que dans les Scrums-de-Scrums niveau global il était question de reporting la plupart du temps. Nous avons rarement eu de vraies discussions dans ce groupe. De plus, de nombreuses personnes en dehors du groupe étaient demandeurs de ce type d'information. Fondamentalement, les équipes veulent savoir ce que font les autres équipes. Du coup nous nous sommes dit que si nous allions nous réunir et passer du temps à s'informer les uns les autres sur ce que

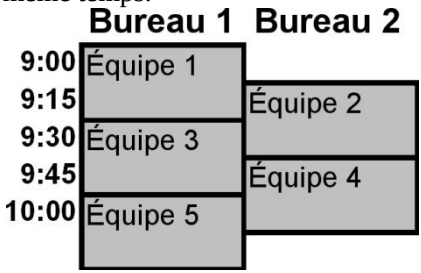
fait chaque équipe, alors pourquoi ne pas laisser venir tout le monde tout simplement.

Comme vous le voyez, la mise en place de scrums-de-scrums est très contextuelle. Il n'y a véritablement pas de solution unique. Je suis parfois choqué par les sociétés qui ont les mêmes réunions abrutissantes et inefficaces chaque semaine (ou tous les jours !), avec des personnes au regard vitreux, fixant le sol, ruminant des rapports sur la situation comme s'ils lisaient un script. Ne soyez pas un zombie Scrum ! Changez quelque chose ! Expérimentez ! Posez-vous constamment des questions entre vous comme "Est-ce que cette réunion apporte vraiment de la valeur ? Pourrait-elle potentiellement apporter plus de valeur ? Que devrions-nous changer pour lui faire apporter plus de valeur ? » La vie est trop courte pour des réunions ennuyantes.

Intercaler les mêlées quotidiennes

Si vous avez beaucoup d'équipes Scrum pour un seul produit, et qu'ils font tous la mêlée quotidienne en même temps, vous avez un problème. Le directeur de produit (et les gens fouineurs comme moi) peuvent seulement assister à une mêlée quotidienne par jour.

Nous demandons alors aux équipes d'éviter d'avoir les mêlées quotidiennes en même temps.



L'exemple de calendrier ci-dessus correspond à la période durant laquelle nous avions des mêlées quotidienne dans des bureaux séparés, plutôt que dans le bureau des équipes. Les réunions durent normalement 15 minutes mais chaque équipe se réserve un espace de 30 minutes au cas où il y aurait besoin de dépasser légèrement.

C'est *extrêmement intéressant* pour deux raisons :

1. Les gens comme le directeur de produit et moi-même pouvons visiter *toutes* les mêlées quotidiennes dans une seule matinée. Il n'y a pas de meilleur moyen d'obtenir une vision réaliste de l'état du sprint, et des menaces clés.

2. Les équipes peuvent visiter les mêlées quotidiennes des autres équipes. Cela n'arrive pas trop souvent, mais de temps en temps deux équipes vont travailler sur une zone similaire, et ainsi quelques membres se rendent visite dans les mêlées pour rester synchronisés.

L'inconvénient c'est moins de liberté pour l'équipe – ils ne peuvent pas choisir le moment qu'ils préfèrent pour la mêlée. Toutefois cela ne s'est pas révélé être un véritable problème pour nous.

Cela m'embête un peu. Le but premier de la mêlée quotidienne est aider les membres de l'équipe à se synchroniser entre-eux. Ils devraient choisir un moment qui leur convient. Garder des gens curieux comme moi dans la boucle – et bien, c'est secondaire. Alors oui, c'est bien d'avoir des mêlées quotidiennes qui ne se superposent pas. Mais ne l'imposez pas aux équipes.

Les équipes de pompiers

Nous avons eu une situation ou un gros produit était incapable d'adopter Scrum parce que les membres de l'équipe passaient beaucoup trop de temps à faire les pompiers, c'est-à-dire à corriger des bugs dans la panique dans leur système livré prématurément. C'était un vrai cercle vicieux, ils étaient si occupés à éteindre les incendies qu'ils n'avaient pas le temps de travailler proactivement pour *prévenir* les feux (c'est-à-dire améliorer la conception, automatiser les tests, créer des outils de surveillance, des outils d'alarme, etc.).

Nous avons géré ce problème en créant une équipe de pompiers dédiée, et une équipe Scrum dédiée.

Le travail de l'équipe Scrum était (avec la bénédiction du directeur de produit) d'essayer de stabiliser le système, et effectivement de prévenir les incendies.
L'équipe de pompiers (nous les appelons «support» en réalité) avait deux boulots :
1) Combattre les incendies.
2) Protéger l'équipe Scrum de toutes les sortes de perturbations, ce qui inclut des choses comme parer des demandes de nouvelles fonctionnalités ad-hoc provenant de nulle part.

L'équipe de pompiers était placée au plus près de la porte ; l'équipe Scrum était placée au fond de la pièce. Ainsi l'équipe de pompiers pouvait en fait *protéger physiquement* l'équipe Scrum des perturbations venant par exemple de vendeurs impatients ou de clients mécontents.

Des développeurs seniors étaient placés dans les deux équipes, de telle sorte qu'une équipe ne soit pas trop dépendante des compétences clés de l'autre équipe.

C'était en fait une tentative de résoudre un problème d'amorçage de Scrum. Comment pouvons nous commencer Scrum si l'équipe n'a aucune chance de planifier son travail plus d'un jour à l'avance ? Notre stratégie a donc été, comme mentionné ci-dessus, de partager le groupe.

Cela a plutôt bien fonctionné. Comme l'équipe Scrum avait reçu du temps pour travailler de manière proactive, ils ont finalement réussi à stabiliser le système. Pendant ce temps l'équipe de pompiers avait complètement abandonné la notion d'être capable de planifier à l'avance, ils travaillaient uniquement de manière réactive, et corrigeaient juste la crise en cours.

J'aurais clairement utilisé kanban pour l'équipe de pompiers, si j'en avais eu connaissance à cette époque.

Bien sûr l'équipe Scrum n'était pas *complètement* à l'abri des perturbations. Fréquemment l'équipe de pompiers devait impliquer des personnes clés de l'équipe Scrum, ou au pire l'équipe en entier.

Néanmoins, après quelques mois, le système était suffisamment stable pour que nous puissions enterrer l'équipe de pompiers et créer des équipes Scrum supplémentaires à la place. Les pompiers étaient plutôt heureux de ranger leurs casques abimés et de rejoindre des équipes Scrum à la place.

C'est un excellent modèle, mais seulement comme stratégie temporaire de gestion de crise. Normalement, vous ne devriez pas avoir besoin d'une équipe de pompiers séparée. Si vous isolez les autres équipes des feux, ils n'apprendront jamais à se prémunir contre les nouveaux ! A la place, si vous avez un problème avec des bugs et des feux survenant de temps en temps (comme à peu près toutes les sociétés), laissez chaque équipe trouver un moyen de les traiter. La plupart des équipes finissent par avoir une sorte de rôle de pompier tournant au sein de l'équipe. Simple et efficace. Et cela leur donne une vraie motivation à écrire du code qui ne prend pas feu !

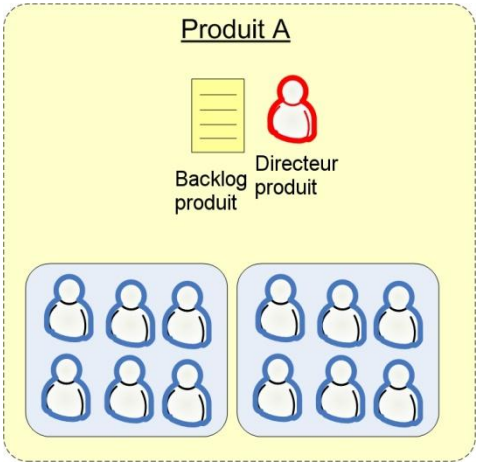
Partager le backlog de produit – ou pas ?

Disons que vous avez un produit et deux équipes Scrum. Combien de backlogs de produit devriez-vous avoir ? Combien de directeurs de produit ? Nous avons évalué trois modèles pour cela. Le choix a un gros effet sur la manière dont les réunions de planning de sprint sont conduites.

Stratégie 1 : un directeur de produit, un backlog

C'est le modèle « Il n'en restera qu'Un ». Notre modèle préféré.

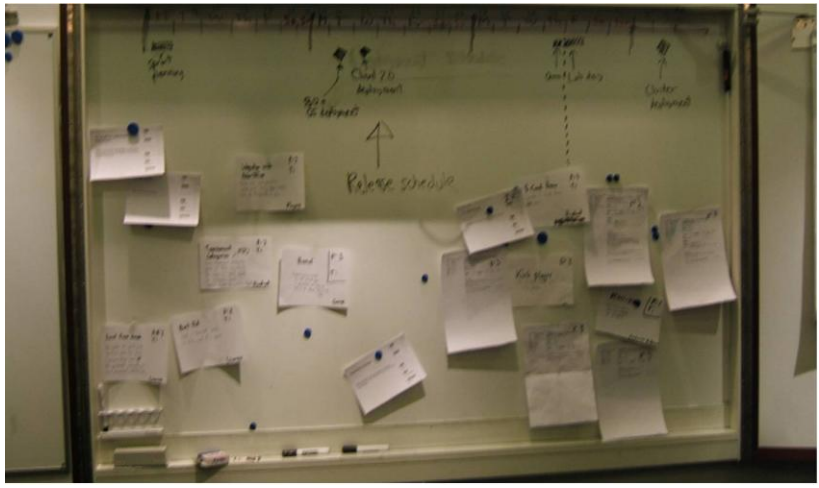
Le bon côté de ce modèle est qu'il laisse les équipes se former à peu près elles-mêmes sur la base des priorités actuelles du directeur de produit. Le directeur de produit peut se concentrer sur *ce dont il a besoin*, et peut laisser les équipes décider comment partager le travail.



Pour être plus concret, voici comment la réunion de planning de sprint fonctionne pour cette équipe :

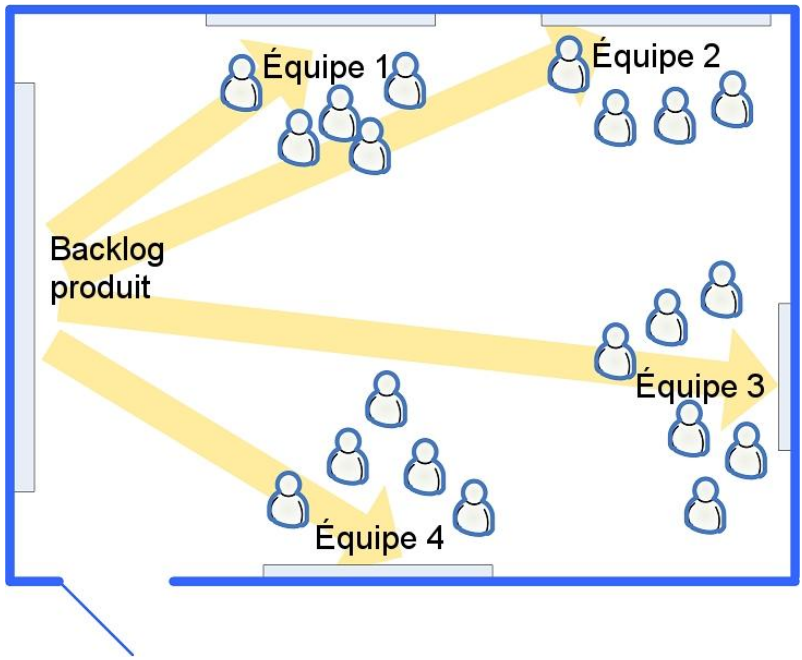
La réunion de planning de sprint a lieu dans un centre de conférence externe.

Juste avant la réunion, le directeur de produit désigne un mur comme le « mur du backlog de produit » et y fixe des histoires d'utilisateur (fiches cartonnées), ordonnées par priorité relative.
Il continue à en ajouter jusqu'à ce que le mur soit rempli, ce qui habituellement constitue plus de travail que possible pour un sprint.



Chaque équipe Scrum sélectionne une section de mur vide pour elle, et y poste son nom d'équipe. C'est leur « mur d'équipe ». Chaque équipe prend alors des histoires sur le mur de backlog de produit, en commençant par les histoires de plus haute priorité, et place les fiches cartonnées sur son propre mur d'équipe.

C'est illustré par le diagramme ci-dessous, avec les flèches jaunes qui symbolisent le flux des fiches depuis le mur de backlog de produit vers les murs des équipes.



Au fur et à mesure de la progression de la réunion, le directeur de produit et les équipes marchandent sur les fiches cartonnées, les déplacent entre les équipes, les déplacent vers le haut ou le bas pour changer les priorités, les découpent en éléments plus petits, etc. Après une heure environ, chaque équipe a une première version candidate de backlog de sprint sur leur mur d'équipe. Après cela, les équipes travaillent indépendamment, et font la décomposition en tâches et l'estimation en temps.

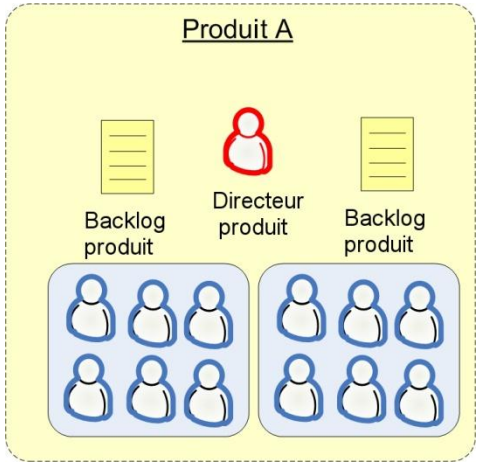
C'est bordélique et chaotique et épuisant, mais également efficace et amusant et social. Quand le temps imparti est écoulé, toutes les équipes ont généralement suffisamment d'informations pour commencer leur sprint.

Ce modèle devient de plus en plus commun ; il est même défini dans certains des plus nouveaux cadres de développement agile, tels que SAFe (Scaled Agile Framework). Récemment chez LEGO, nous avons organisé un événement de planification de deux jours avec plus de 130 personnes ! Parfois, cette technique de planification est utilisée comme une mesure temporaire pour 6 mois environ, jusqu'à ce que l'on trouve une structure d'équipe et une architecture qui donne la possibilité aux équipes de faire des réunions de planning de sprint de manière plus indépendante.

Stratégie 2 : un directeur de produit, plusieurs backlogs

Dans cette stratégie, le directeur de produit maintient *plusieurs* backlogs de produit, un par équipe. Nous n'avons pas vraiment essayé cette approche, mais nous en avons été proches. C'est en fait notre plan de secours au cas où la première approche échoue.

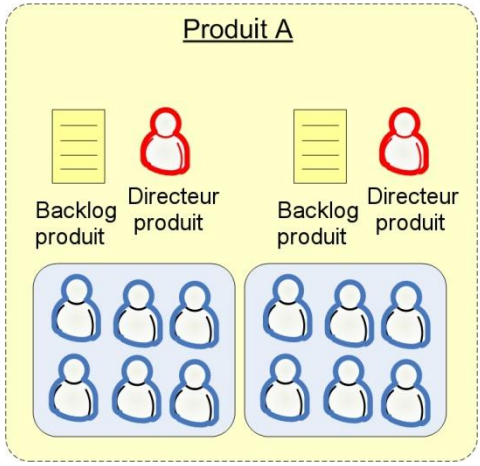
La faiblesse de cette stratégie est que le directeur de produit assigne les histoires aux équipes, un travail que les équipes sont probablement plus qualifiées à faire elles-mêmes.



Typiquement, le product owner devient un goulot d'étranglement dans ce cas.

Stratégie 3 : plusieurs directeurs de produit, un backlog par directeur

C'est comme la seconde stratégie, un backlog de produit par équipe, mais avec un *directeur de produit* par équipe également !



Nous n'avons jamais fait cela, et nous ne le ferons probablement jamais.

Si les deux backlogs de produit impliquent la même base de code, cela va probablement causer de sérieux conflits d'intérêt entre les deux directeurs de produit.

Si les deux backlogs de produit impliquent des bases de code séparées, alors cela revient à partager le produit total en sous-produits séparés, et à les gérer indépendamment. Cela signifie que nous sommes revenus à la situation « une équipe un produit », ce qui est élégant et facile.

C'est le modèle utilisé chez Spotify. Chacune des 70+ équipes a son propre product owner et backlog produit. En fait, dans certains cas, nous avons un bout de Stratégie 1 en cours (un backlog partagé par plusieurs équipes). Mais pour la grande partie, chaque équipe a sont propre product owner et backlog. L'avantage est que nous avons rarement besoin de grandes réunions de planification. L'inconvénient est que nous devons fournir beaucoup d'effort dans le découplage de l'architecture pour rendre cela possible. Comme dans tout, il y a toujours des avantages et des inconvénients. Alors continuez à… euh… (j'allais dire « continuez à expérimenter » mais je l'ai déjà dit tellement de fois… je ne voudrais pas me répéter, n'est ce pas ?)

Branches de code

Avec plusieurs équipes qui travaillent sur la même base de code, il est inévitable d'avoir à gérer des branches de code dans le système de gestion de configurations. Il y a beaucoup de livres et d'articles sur la manière de gérer plusieurs personnes travaillant sur la même base de code, donc je n'irais pas dans les détails ici. Je n'ai rien de nouveau ou de révolutionnaire à ajouter. Toutefois je vais résumer quelques unes des plus importantes leçons que nos équipes ont apprises jusque là.

- Soyez strict et gardez la ligne principale (ou tronc) dans un état cohérent. Cela signifie, au minimum, que tout devrait compiler et que tous les tests unitaires devraient passer. Il devrait être possible de créer une version livrable fonctionnelle à *n'importe quel moment.* De préférence, le système de build en continu devrait construire et déployer automatiquement sur un environnement de test toutes les nuits.
- Etiquetez chaque version livrée. Chaque fois que vous livrez une version pour les tests d'acceptation ou pour la production, assurez-vous qu'il y a une étiquette de version sur votre ligne principale, étiquette qui identifie exactement ce qui a été livré. Cela signifie que vous pouvez, à n'importe quel point dans le futur, revenir en arrière et créer une branche de maintenance depuis cette étiquette.

- Créez de nouvelles branches uniquement quand c'est nécessaire. Une bonne règle empirique est de créer une nouvelle branche de code *uniquement* quand vous ne pouvez pas utiliser une ligne de code existante sans violer la politique de cette ligne. En cas de doute, ne créez pas de branche. Pourquoi ? Parce que chaque branche active est couteuse en administration et complexité.
- Utilisez les branches principalement pour séparer *différents cycles de vie.* Vous pouvez décider ou non d'avoir chaque équipe Scrum sur a propre ligne de code. Mais si vous mélangez des corrections à court terme avec des changements à long terme dans la même ligne de code, vous trouverez très difficile de livrez les corrections à court terme !
- Synchronisez souvent. Si vous travaillez sur une branche, synchronisez avec la ligne principale chaque fois que vous quelque chose qui compile. Tous les matins quand vous arrivez au travail, synchronisez depuis la ligne principale sur votre branche, de telle sorte que votre branche soit à jour par rapport aux modifications des autres équipes. Si fusionner c'est l'enfer, acceptez simplement le fait que cela aurait été bien pire d'attendre.

Ouais, ce conseil est toujours fortement d'actualité (sauf pour tagguer souvent – vous avez ça automatiquement avec la plupart des systèmes aujourd'hui). Avec les systèmes de contrôle de version modernes (comme GIT), il n'y a plus d'excuses pour ne pas committer souvent, gardez le tronc propre, livrez souvent, et gardez les branches éphémères. J'ai écrit un article à ce sujet s'intitulant « Contrôle de Version pour Equipes Agiles Multiples » : http://www.infoq.com/articles/agile-version-control

Je trouve cela marrant que les systèmes comme GIT sont présentés comme des systèmes de contrôle de version « décentralisés », alors que dans presque tous les projets il y a toujours un repo central et une branche master sur laquelle tout le monde pousse et où les releases sont faites. Pareil mais différent. :o)

Rétrospectives multi-équipes

Comment faisons nous les rétrospectives de sprint quand plusieurs équipes travaillent sur le même produit ?

Immédiatement après la démonstration du sprint, après les applaudissements et le mélange général, chaque équipe part dans une pièce réservée pour elle, ou pour un endroit confortable à l'extérieur. Elles font leurs rétrospectives à peu près

comme je l'ai décrit page 82 « Comment nous faisons les rétrospectives de sprint ».

Durant la réunion de planning de sprint (à laquelle toutes les équipes participent, puisque nous faisons des sprints synchronisés pour chaque produit), la première chose que nous faisons est de laisser un porte-parole par équipe faire un résumé des points clés de leur rétrospective. Cela prend environ 5 minutes par équipe. Ensuite nous avons une discussion ouverte pendant 10 à 20 minutes. Puis nous faisons une pause. Ensuite nous commençons le planning de sprint proprement dit.

Nous n'avons pas essayé d'autres manières, ceci marche suffisamment bien. Le plus gros inconvénient est qu'il n'y a pas de temps libre après la rétrospective, et avant le planning. (voir page 93 « Relâcher le rythme entre les sprints »)

Exactement ! Et c'est un bien gros désavantage ! Alors aujourd'hui, dans un scénario multi-équipe avec dépendances, je préfère faire des résumés de rétrospectives inclus dans la véritable rétrospective. Ainsi, chaque équipe s'en va et effectue sa propre rétro, et puis après une heure environ nous nous revoyons encore dans une grande salle. Chaque équipe résume les résultats principaux de leur rétro, et liste leurs plus importants impédimentas non résolus. Après avoir écouté chaque équipe, il apparaît alors généralement clair que l'impédimenta X est notre plus gros impédimenta inter-équipe (ex : « une définition de terminé inconsistante entre équipes » ou « le tronc n'arrête pas de casser »). C'est une opportunité parfaite pour faire un mini-atelier autour de ça, ou de trouver des volontaires (comme une personne de chaque équipe, plus le manager) pour s'en aller et trouver une solution. Parfois, ils trouvent en seulement quelques heures, alors c'est utile de faire la réunion de planning de sprint le jour suivant.

Pour les produits à une seule équipe, nous ne faisons pas ce résumé de rétrospective durant la réunion de planning de sprint. Cela n'est pas nécessaire, puisque tout le monde était présent lors de la rétrospective du sprint.

Le contexte est roi, comme dit le dicton. Mais c'est particulièrement vrai avec la mise à l'échelle ! Il n'y a vraiment pas de solution unique – à peu près toutes les approches peuvent obtenir une très grande réussite ou subir un échec total selon le contexte.

J'ai toutefois trouvé de forts dénominateurs communs : travailler en équipes multidisciplinaires, visualiser les choses, livrer souvent, impliquer de vrais utilisateurs, automatiser vos tests et votre processus de livraison, et expérimenter beaucoup. La base des principes agiles, vraiment.

16

Comment nous gérons les équipes géographiquement dispersées

Que se passe-t-il lorsque des membres de l'équipe sont dans des lieux géographiques différents ? Le plus gros de la « magie » de Scrum et XP s'appuie sur des équipiers colocalisés qui collaborent étroitement en programmant par paires et en se réunissant chaque jour.

Nous avons des équipes séparées géographiquement les unes des autres, ainsi que des équipiers qui travaillent de chez eux de temps en temps.

Notre stratégie est alors assez simple. Nous employons toutes les ficelles imaginables pour maximiser la bande passante utilisée par les équipiers séparés physiquement, pour communiquer entre eux. Je ne parle pas seulement de bande passante en termes de Mbits/seconde (bien que celle-ci soit évidemment importante également). Je parle de bande passante en termes de communication, au sens plus large :

- La capacité à programmer par paires.
- La capacité à se rencontrer en tête-à-tête lors de la mêlée quotidienne.
- La capacité à discuter en tête-à-tête à tout moment.
- La capacité à se rencontrer physiquement et en dehors du contexte du travail.
- La capacité à réunir toute l'équipe, de façon impromptue.
- La capacité à accéder à la même vue du Sprint Backlog, du Sprint Burndown, du Backlog des produits et d'autres sources d'information.

Certaines des mesures que nous avons implémentées (ou que nous sommes en train d'implémenter, elles ne sont pas encore toutes opérationnelles) sont :

- Une webcam et un casque avec microphone à chaque station de travail.
- Des salles de conférence équipées pour la communication à distance avec des webcams, des microphones de conférence, des ordinateurs allumés en permanence, des logiciels de partage des postes de travail, etc.

- Des « fenêtres distantes ». De grands écrans dans chaque lieu, montrant en permanence une vue des autres lieux. Un peu comme une fenêtre virtuelle entre deux services. Vous pouvez vous faire des signes. Vous pouvez voir qui est à son poste et qui parle à qui. Le but est de créer le sentiment que « hé, nous sommes tous dans le même bateau ».
- Des programmes d'échange, lors desquels des personnes provenant des divers lieux se rendent visite régulièrement.

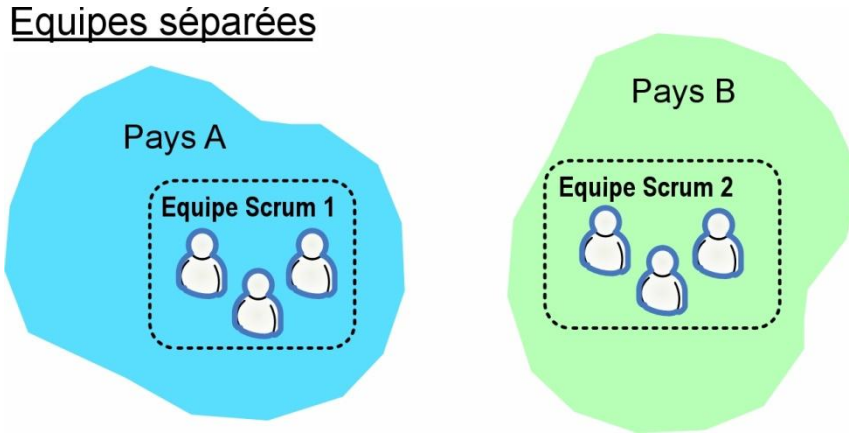
En utilisant ces techniques ainsi que d'autres, nous commençons lentement mais sûrement à nous habituer à organiser des réunions de planification du sprint, des démonstrations, des rétrospectives, des mêlées quotidiennes, etc. avec les équipiers dispersés géographiquement.

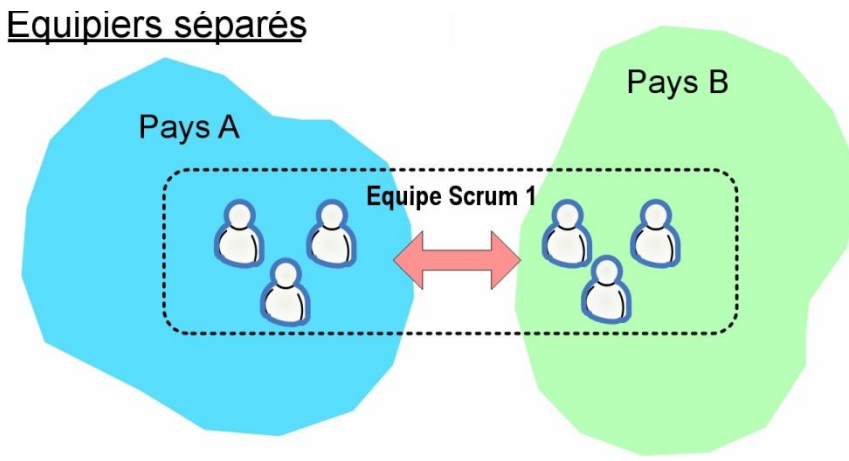
Les équipes distribuées sont partout (jeu de mots). La plupart des projets open-source sont bâtis sur des équipes totalement distribuées, alors il n'y a aucun doute que cela est possible. Néanmoins, rien ne peut battre la productivité d'une petite équipe multidisciplinaire installée ensemble dans la même salle, focalisée à 100% sur un unique objectif partagé. Alors la première priorité devrait toujours être d'essayer de parvenir à ce contexte, ou de s'en approcher le plus possible. Si vous ne pouvez toujours pas éviter d'avoir des membres d'équipe dans différents endroits, les outils et techniques listées au-dessus sont une excellente façon de limiter les dégâts. Alors ne soyez pas avare, prenez les meilleurs outils que votre argent peut acheter ! Vous ne serez jamais aussi productif qu'une équipe colocalisée, mais vous pourriez vous en rapprocher.

Comme d'habitude, c'est une histoire d'expérimentation. Inspecte => adapte => inspecte => adapte => inspecte => adapte => inspecte => adapte => inspecte => adapte

Nous avons plusieurs équipes « offshore » et expérimentons depuis un certain temps des façons de gérer la situation avec Scrum.

Il existe ici deux stratégies principales : des équipes séparées et des équipiers séparés.





La première stratégie, des équipes séparées, est un choix incontestable. Cependant, nous avons commencé par la seconde stratégie, des équipiers séparés. Pour cela, il y a plusieurs raisons.

1. Nous voulons que les équipiers se connaissent bien entre eux.

2. Nous voulons une excellente infrastructure de communication entre les deux lieux et voulons fortement inciter les équipes à la mettre en place.
3. Au début, l'équipe « offshore » est trop petite pour constituer une équipe Scrum efficace à elle seule.
4. Nous n'envisagerons la création d'équipes offshore qu'au terme d'une période d'échange intense des connaissances.

A long terme, il se peut que nous nous dirigions vers la stratégie des « équipes séparées ».

C'est une stratégie plutôt bonne globalement. Dans le long terme, vous voudrez vraiment des équipes colocalisées, parce qu'une équipe distribuée n'atteindra jamais le même niveau de « travail d'équipe ». Alors même si vous avez initialement une équipe distribuée, continuez à chercher des moyens pour créer des équipes séparées, distinctes dans chaque lieu. Vous aurez toujours besoin de gérer les dépendances et les problèmes de communication entre les deux équipes, mais c'est un problème plus facile à traiter. En plus de l'amélioration de la communication de tous les jours, un des gains principaux est la motivation. Avoir ses équipiers dans la même salle est amusant ! Et les personnes motivées construisent de meilleurs trucs, plus vite.

Équipiers travaillant de la maison

Le travail accompli à la maison peut être quelquefois très bon. Il arrive d'accomplir plus de programmation en un jour à la maison qu'en toute une semaine au bureau. Du moins si vous n'avez pas d'enfants :o)

Pourtant, l'un des fondements de Scrum est que toute l'équipe devrait être colocalisée physiquement. Alors que fait-on ?

En gros nous laissons les équipes décider d'elles-mêmes quand et à quelle fréquence il est acceptable de travailler de la maison. Certains équipiers travaillent régulièrement de la maison afin d'éviter les longs trajets. Nous encourageons cependant les équipes à être colocalisées physiquement « la plupart » du temps.

Lorsque les équipiers travaillent de la maison, ils participent à la mêlée quotidienne *via* un appel Skype (quelquefois avec vidéo). Ils sont en ligne toute la journée *via* la messagerie instantanée. Pas aussi bien que d'être dans la même pièce, mais suffisant.

Il y a beaucoup de systèmes de télétravail vraiment cool et bon marché disponibles de nos jours. Par exemple, regardez Double Robotics sur http://doublerobotics.com. Ils vendent un produit qui est en fait un iPad sur un bâton à roulettes. C'est comme être sur un appel Skype, mais vous pouvez bouger autour aussi ! Je l'utilise de plus en plus souvent ; on a vraiment l'impression de se téléporter dans un autre lieu !

Nous avons une fois essayé le concept de désigner le mercredi comme étant le *jour de focalisation*. En gros cela voulait dire « si vous aimeriez travailler de la maison, c'est très bien, mais faites-le le mercredi. Et restez en contact avec votre équipe ». Cela a assez bien fonctionné avec l'équipe sur laquelle nous l'avons essayé. En général la plupart des équipiers restaient chez eux le mercredi et étaient productifs, tout en collaborant assez bien. Puisque cela ne durait qu'un jour, les équipiers restaient à peu près synchronisés entre eux. Mais pour une raison inconnue, ce concept n'a pas vraiment pris dans les autres équipes.

En général, le fait que des personnes travaillent de chez eux ne nous a pas vraiment posé de problème.

Une des idées clés dans Scrum est l'équipe auto-organisée. Son importance ne peut pas être exagérée. On devrait donner à l'équipe le plus de responsabilité possible, incluant des choses comme les heures de travail et les politiques de télétravail. L'auto-organisation est la clé de la créativité, de la motivation, de l'innovation, et plein d'autres Bonnes Choses ! Certains managers ont peur que les équipes abusent de cette confiance, mais j'ai rarement vu cela arriver en pratique. Tant que l'équipe est clairement responsable pour le produit qu'elle délivre, elle tend à agir de manière responsable.

17

Pense-bête du Scrum Master

Dans cet ultime chapitre, je vais vous montrer notre « pense-bête » du Scrum master, qui inventorie les routines administratives les plus courantes pour nos Scrum Masters. Des choses qui s'oublient facilement. Nous passons sur les évidences telles que « supprimer les obstacles rencontrés par l'équipe ».

Aussi, jetez un oeil sur la Checklist Scrum. https://www.crisp.se/gratis-material-och-guider/scrum-checklist

C'est officiellement la checklist non officielle de Scrum, mais elle a été tellement utilisée que vous pourriez dire qu'elle est non officiellement devenue la checklist officielle de Scrum. :o)

Début du Sprint

- Après la réunion de planification du Sprint, créer une page d'information du Sprint.
  - Ajouter un lien à cette page depuis le tableau de bord sur le wiki.
  - Imprimer la page et l'afficher sur le mur près de l'équipe et d'un lieu de passage.
- Envoyer un e-mail à chacun pour annoncer qu'un nouveau Sprint a démarré. Y inclure le but du Sprint ainsi qu'un lien vers la page d'information du Sprint.
- Mettre à jour le document des statistiques du Sprint. Ajouter votre vélocité estimée, la taille de l'équipe, la durée du Sprint, etc.

Tous les jours

- S'assurer que la Mêlée quotidienne commence et finit à l'heure.

- S'assurer que les histoires sont ajoutées/supprimées du Sprint Backlog autant que nécessaire pour respecter le planning du Sprint.
    - S'assurer que le directeur de produit est informé de ces modifications.
- S'assurer que l'équipe tient le Backlog et le Burndown du Sprint à jour.
- S'assurer que les problèmes/obstacles sont résolus ou communiqués au directeur de produit et/ou au responsable du développement.

Fin du Sprint

- Organisez une démonstration publique à l'issue du Sprint.
- Tout le monde devrait être informé de la tenue de la démonstration un ou deux jours avant.
- Organisez une rétrospective du Sprint en présence de toute l'équipe et du directeur de produit. Invitez également le responsable du développement afin qu'il puisse propager les leçons apprises.
- Mettez à jour le document des statistiques du Sprint. Ajoutez-y la vélocité effective ainsi que les points clés de la rétrospective.

Bonne petite checklist. Bien qu'avec le temps, en tant que Scrum master, essayez de vous rendre redondant. Coachez l'équipe pour qu'elle fasse ces choses sans vous. Même si vous n'arrivez pas à vous rendre redondant, le seul acte d'essayer vous emmènera à faire de Bonnes Choses. Certains Scrum masters finissent dans un rôle plus de l'ordre de l'administrateur Scrum ou de l'esclave Scrum, tellement ils sont désireux de faire plaisir à l'équipe. Si l'équipe s'appuie trop sur vous, alors vous entravez effectivement leur capacité à s'auto-organiser. Au début, ce n'est pas grave si l'équipe est novice à Scrum et a besoin de votre aide. Mais avec le temps, vous devriez doucement vous retirer de l'administration des trucs et donner de plus en plus de responsabilité à l'équipe. Cela préserve du temps pour vous pour chasser les impédimentas, ou pour faire d'autres trucs – hors Scrum Master - comme coder, tester, etc.

18

Mot de la fin

Eh bien ! Je n'aurais jamais cru que ça deviendrait aussi long.

J'espère que ce papier vous a donné des idées utiles, que Scrum soit nouveau pour vous ou que vous soyez un vétéran chevronné.

Parce que Scrum doit être façonné différemment pour chaque environnement, il est difficile de débattre objectivement des meilleures pratiques en général. Néanmoins, je suis intéressé par vos remarques. Dites-moi en quoi votre approche diffère de la mienne. Donnez-moi des idées d'amélioration !

N'hésitez pas à me contacter à **henrik.kniberg@crisp.se**.

Oh, CELA explique pourquoi je reçois autant d'emails… j'apprécie vraiment le feedback, mais ne vous sentez pas offensé si je ne réponds pas. J'essaie d'avoir une vie aussi, parfois. :o)

Je garde aussi un œil sur **scrumdevelopment@yahoogroups.com**.

Si vous avez aimé ce livre, vous aurez peut-être envie de jeter un œil à mon blog de temps en temps. J'espère y ajouter des articles sur Java et le développement informatique agile.
http://blog.crisp.se/henrikkniberg/

Il y a BEAUCOUP d'articles, de vidéos et de trucs sur ce blog maintenant ! Pas juste mes trucs, mais de mes collègues de Crisp et de partenaires variés et d'amis. C'est une mine d'or, honnêtement !

Je suis également plutôt actif sur Twitter de temps en temps (@henrikkniberg).

Ah oui, et n'oubliez pas…
Ce n'est jamais qu'un boulot, non ?

Lectures recommandées

Voici quelques livres qui m'ont donné beaucoup d'inspiration et d'idées. Hautement recommandés !

Euh… Je devrais vraiment mettre ça à jour. J'ai lu tellement de livres intéressants depuis 2007 ! Mais ça devra être pour un post séparé un jour. #cliffhanger

A propos de l'auteur

Henrik Kniberg (henrik.kniberg@crisp.se) est un consultant chez Crisp à Stockholm (www.crisp.se), spécialisé dans Java et le développement informatique Agile.

Aujourd'hui, je suis plus comme une sorte de consultant en management, chercheur organisationnel, ou coach lean/agile. En fait, j'aide à refactorer, débugger, et optimiser les entreprises. Je code toujours de temps en temps, toutefois, dans mon temps libre. C'est juste trop amusant !

Dès l'apparition des premiers livres sur XP et du manifeste agile, Henrik adopta les principes agiles et essaya d'apprendre à les appliquer de façon efficace dans différents types d'organisations. En tant que cofondateur et Directeur Technique de Goyada entre 1998 et 2003, il eût amplement l'opportunité d'expérimenter avec le développement dirigé par les tests ainsi que d'autres pratiques agiles, alors qu'il construisait et manageait une plateforme technique et une équipe de développement de 30 personnes.

Fin 2005, Henrik effectua une mission en tant que responsable du développement dans une société suédoise dans le domaine du jeu. La société était en situation de crise avec des problèmes organisationnels et techniques urgents. Par l'utilisation de Scrum et XP comme outils, Henrik aida la société à sortir de sa crise en implémentant des principes agiles à tous les niveaux de la société.

Un vendredi de novembre 2006, Henrik était alité avec de la fièvre et décida de prendre quelques notes sur ce qu'il avait appris pendant l'année passée. Mais une fois qu'il eût commencé à écrire, il ne pouvait plus s'arrêter et après avoir tapé sur son clavier et dessiné pendant trois jours, les notes initiales s'étaient transformées en un article de 80 pages intitulé « Scrum et XP depuis les Tranchées », qui éventuellement devint ce livre.

Je suis toujours surpris d'avoir réussi à écrire tout ce fichu truc en un weekend ! D'autres auteurs me détestent pour ça.

Henrik adopte une approche holistique et apprécie de jouer différents rôles tels que manager, développeur, Scrum Master, professeur et entraîneur. Aider les sociétés à développer des applications et des équipes excellentes le passionne, adoptant tout rôle qu'il juge nécessaire.

Henrik a grandi à Tokyo et vit maintenant à Stockholm avec son épouse Sophia et leurs deux enfants.

Même femme et enfants, juste plus. Des enfants, bien évidemment :o)

Durant son temps libre, il compose de la musique et joue de la basse et du clavier avec des groupes locaux.

Pour plus d'information, visitez http://www.crisp.se/henrik.kniberg