Lean depuis les Tranchées

Un exemple d'application de Kanban dans le cadre d'un gros projet logiciel

Date : 01/08/2011
Auteur : Henrik Kniberg
Version : draft0.9

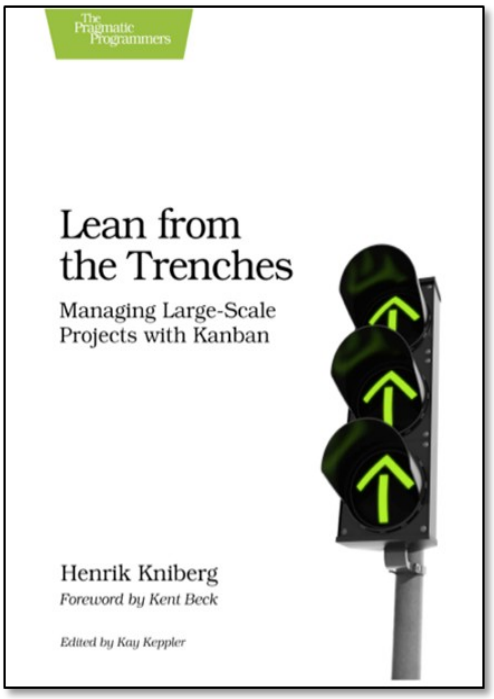
Version draft traduite le 20/03/2012 par :
Claude Aubry, Sylvain Fraïssé, Nicolas Mereaux,
Antoine Vernois et Fabrice Aimetti

NOTE – cet article constitue un premier jet du livre "Lean depuis les Tranchées – Piloter de Gros Projets Avec Kanban"

La version du livre est bien meilleure. Vraiment !

- Des chapitres supplémentaires tels que “Agile & Lean en un mot" et "Les diagrammes de causes et effets"
- Des photos plus nombreuses et de meilleure qualité
- Beaucoup de clarifications et de réponses aux questions fréquentes.

Achetez le livre ou sa version ebook ici :
http://pragprog.com/book/hklean/lean-from-the-trenches



Cette version draft est gratuite. Je l'ai rendue disponible pour ceux qui ont un budget serré et qui ne peuvent pas s'offrir le livre, ou qui souhaitent parcourir le draft pour se faire une idée générale de la portée du livre avant de l'acheter.

Profitez-en !
/Henrik Kniberg, le 19/12/2011

0. Glossaire

Ajouté par les traducteurs pour permettre aux lecteurs de comprendre les choix de traduction de certains termes anglais.

Agile	Agilité
Daily stand-up	Mêlée quotidienne
Epic	Epopée
Feature	Fonctionnalité
Feature Team	Équipe de développement
Feedback	Feedback
Lean	Lean
Lean product development	Développement Lean de produits
Lean Software Development	Lean Software Development
Manager	Manager
Story	Histoire
WIP – Work In Progress	TAF – Travail À Finir

1. Introduction

Beaucoup d'entre nous ont entendu parler du Lean Software Development, de Kanban et d'autres termes très à la mode. Mais à quoi ressemble réellement ces choses lorsqu'on les met en pratique ? Comment s'appliquent-elles à un gros projet de 60 personnes développant un système vraiment complexe ?

Je ne peux pas vous dire comment faire, puisque chaque contexte est particulier. Mais je vais vous dire comment nous l'avons fait (dans les grandes lignes, un hybride de Scrum, XP et Kanban), et peut-être que certaines des solutions et leçons apprises pourront se révéler précieuses dans votre propre contexte.

Ne vous attendez pas à lire de la théorie sur le Lean ou le Kanban dans ce livre. Bon, si, quand même un peu. Pour le reste, je vous renvoie à la dizaine de livres écrits sur le sujet. Ce papier est essentiellement un exemple tiré de la vie réelle.

Audience ciblée

Ce livre s'adresse principalement aux managers, coachs et autres agents du changement au sein d'une organisation développant du logiciel. Toutefois, certaines parties de ce livre se révèleront probablement utiles à toute personne intéressée par le développement logiciel, le développement Lean de produits, ou autres techniques de collaboration, quels que soient son rôle et son secteur d'activité.

Avertissements

Je ne prétends pas que notre façon de travailler est parfaitement Lean. Le Lean est une direction, pas un lieu, il est entièrement centré sur l'amélioration continue. Il n'y pas de définition claire du Lean, mais la plupart des pratiques que nous mettons en oeuvre sont fondées sur les principes du développement Lean de produits enseignés par Mary Poppendieck, David Anderson et Don Reinertsen. Ces principes Lean se marient assez bien avec les principes de l'Agilité :o)

Autre chose. Je suis, comme tout être humain, subjectif. Vous verrez ce projet avec mes yeux, c'est-à-dire celui d'un coach à temps partiel sur 6 mois. J'ai peut-être oublié certains éléments importants et probablement mal compris d'autres. Mon objectif n'est pas de restituer une image correcte à 100% mais juste de vous donner une idée générale de ce que nous avons fait et de ce que nous avons appris jusqu'ici.

Remerciements

Beaucoup de personnes ont contribué au fait que ce livre existe, merci à tous ! Je souhaiterais plus particulièrement remercier Håkan Rydman pour m'avoir amené dans ce projet et avoir été l'agent du changement en interne, ainsi que Tomas Alsterlund pour m'avoir fourni un énorme soutien managérial et nous avoir maintenu alignés sur l'objectif du projet.

Je souhaiterais également remercier tous ceux qui ont participé au projet, pour y avoir mis tout leur coeur, et m'avoir aidé à mettre en oeuvre le processus de changement. J'ai été surpris par les compétences, la créativité et l'énergie déployées par cette équipe projet !

Enfin, je souhaiterais remercier ma femme Sophia qui m'a permis de rester assez concentré et productif (ce qui n'est pas facile avec 4 enfants en bas âge à la maison...) pour finir la première version de ce livre en quelques jours plutôt qu'en quelques mois.

2. À propos du projet

La RPS (rikspolisstyrelsen) est la police nationale suédoise, et le produit que nous avons développé est un nouveau système numérique dédié aux enquêtes de police nommé PUST ("Polisens mobila Utrednings STöd"). L'idée principale est d'équiper chaque voiture de police d'un petit ordinateur portable muni d'une connexion internet mobile et d'une application web lui permettant de gérer sur le terrain tout le travail d'enquête lié à une affaire judiciaire.

Supposons que la police arrête un conducteur ivre. Auparavant, le policier devait noter toutes les informations sur papier, puis se rendre au poste de police, faire un tas de paperasses administratives pour finalement passer le dossier à un autre enquêteur qui ferait avancer l'affaire. Cela prenait généralement des mois.

Avec PUST, la police saisit toutes les informations directement sur le portable, qui est en permanence relié à tous les systèmes concernés. Le dossier est classé en quelques jours voire en quelques heures.

Le système a été déployé nationalement en avril 2011 ; il a bénéficié de toute l'attention des médias : le produit a été cité dans les grands journaux, à la télévision et à la radio. Il a reçu un accueil très positif de la part des parties prenantes.

Le temps moyen de traitement pour la plupart des infractions mineures est six fois plus rapide, du coup la police peut passer plus de temps sur le terrain et moins au poste de police. Non seulement cela a réduit le nombre de crimes mais c'est aussi beaucoup plus motivant pour les policiers qui préfèrent faire leur travail de policiers, pas de la paperasserie !

Par ailleurs, et de façon surprenante, il y a eu peu de demandes de support ou de rapports d'anomalies, comparé aux autres projets de complexité et dimension similaires.

PUST est un système complexe car :

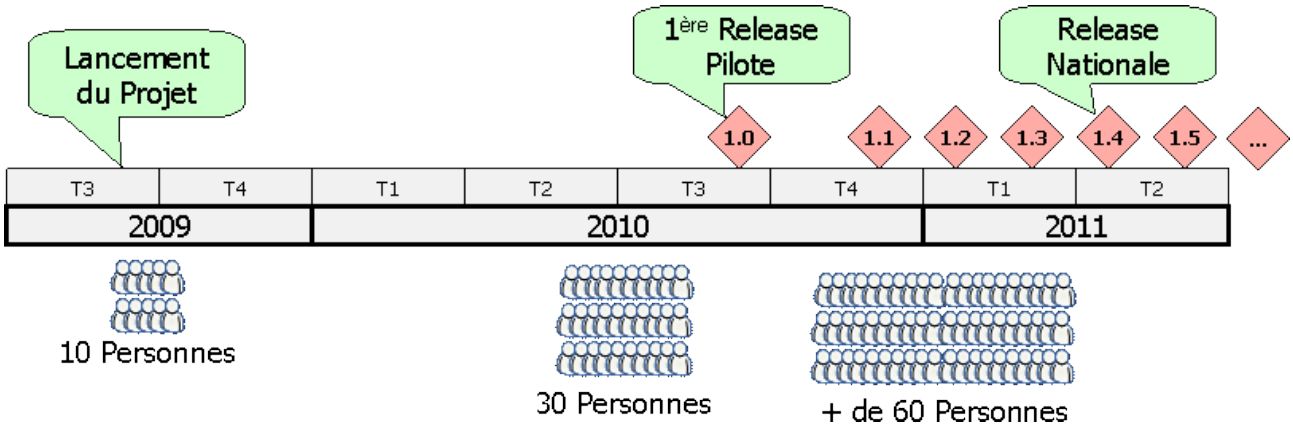
- il doit s'interfacer avec un très grand nombre de systèmes existants,
- il doit être très ergonomique, puisque la police l'utilise en temps réel pendant les interrogatoires,
- il doit être très sécurisé,
- il doit respecter un très grand nombre de lois et règlements compliqués.

C'était un projet très important pour la RPS. Les bénéfices attendus étaient énormes, à l'image des coûts et des risques. Le côté positif de la situation a été qu'on nous a permis d'essayer de nouvelles –et plus efficaces– manières de travailler, qui ont finalement conduit à l'écriture du présent livre.

PUST est un maillon du changement culturel en cours au sein de la police suédoise, dans le cadre d'une initiative Lean menée à l'échelle de l'organisation. Il était donc tout à fait sensé d'appliquer aussi les principes du Lean au processus de développement logiciel :o)

3. Déroulement

Le développement a commencé aux alentours de septembre 2009. La première version en production (un pilote) est arrivée un an plus tard, suivie d'une version tous les deux mois.



Un an pour une première version pourrait sembler long à des agilistes, mais comparé à d'autres projets gouvernementaux d'un périmètre et d'une complexité comparables, c'est extrêmement court ! Certains projets gouvernementaux peuvent prendre jusqu'à 7 ans avant la première mise en production !

La version 1.0 était une version pilote, la 1.4 était la version principale étendue à l'ensemble du pays. Livrer une version en production tous les deux mois est aussi inhabituel, beaucoup d'organisations gouvernementales livrant seulement une ou deux fois par an.

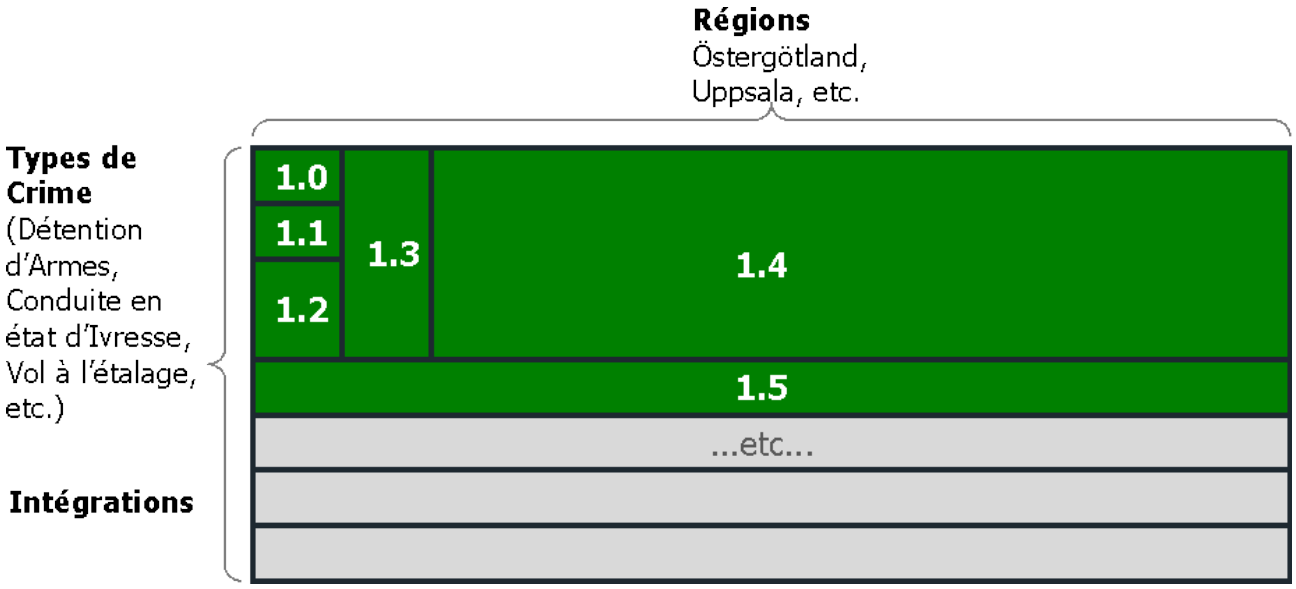
Le projet a commencé avec une dizaine de personnes au 3ème trimestre 2009. Il s'est agrandi, comptant une trentaine de personnes à la mi-2010 et plus de 60 au 4ème trimestre 2010.

Tous ces facteurs – les cycles de livraison rapides et le passage agressif à grande échelle – nous ont forcés à faire évoluer rapidement l'organisation et le processus de développement.

4. Comment nous avons découpé le mastodonte

La clé pour minimiser les risques dans les gros projets est de trouver une manière de « découper le mastodonte », c'est-à-dire livrer le système par petits incréments au lieu d'attendre que tout soit fini pour faire une intégration de type big bang.

Nous avons fait une décomposition sur deux dimensions : la localisation géographique et le type de crimes.



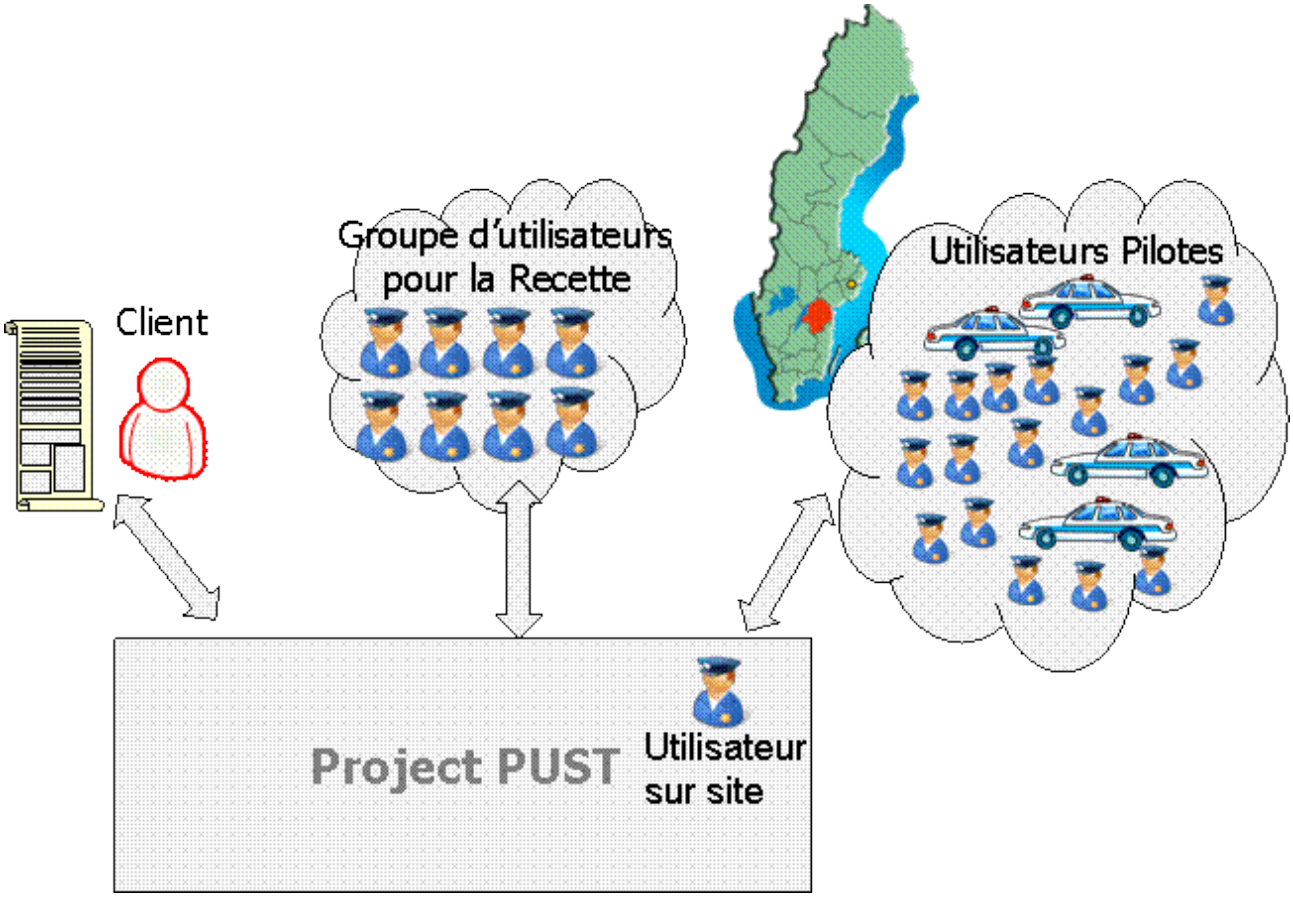
- **1.0-1.2** : Versions pilotes pour une unique région – Östergötland – et ne supportant qu'un nombre réduit de types de crimes de droits communs tels que la conduite en état d'ivresse ou la possession d'armes blanches. Les autres types de crimes sont encore gérés, à l'ancienne, manuellement.

  Par la suite, nous avons amélioré la stabilité et ajouté des types de crimes supplémentaires.
- **1.3** : Prise en compte d'une nouvelle région – Uppsala.
- **1.4** : Extension au reste de la Suède. Cela a été la version « principale ».
- **1.5** : Ajout de types de crimes supplémentaires et nouvelles intégrations à divers systèmes.

En plus des livraisons de fonctionnalités tous les deux mois, nous faisions des livraisons de type « patch » presque toutes les semaines avec des améliorations mineures et des corrections d'anomalies sur les fonctionnalités existantes.

5. Comment nous avons impliqué les clients et utilisateurs

Considérons le projet comme une « boîte noire » et voyons ses interactions avec les clients et les utilisateurs :



Une personne tenait le rôle de « client » principal du projet et élaborait la liste de fonctionnalités, d'un niveau assez élevé que nous appelions nos « domaines de fonctionnalités ». Elles correspondaient à peu près à ce que les agilistes appellent des épopées.

Sur le site, il y avait également un utilisateur, un vrai utilisateur, qui était dans le bâtiment avec les équipes de développement. Au début il était là environ une fois par semaine, mais au cours des dernières étapes nous avions des utilisateurs sur site presque tous les jours, selon un planning tournant. Ces utilisateurs sur site étaient là pour donner leur feedback, voir les démos, répondre aux questions, etc...

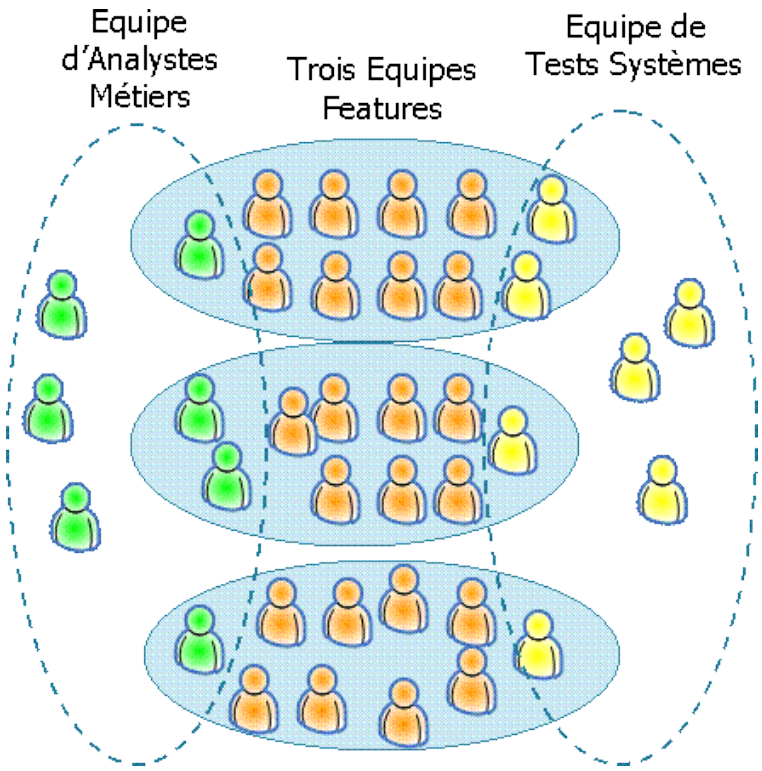
Une semaine avant la fin de chaque release, un groupe de personnes venait pour les tests d'acceptation, typiquement une dizaine d'officiers de police et autres vrais utilisateurs. Ce groupe passait quelques jours à essayer la dernière version candidate et donnait son feedback. Habituellement ce système marchait si bien qu'au moment d'atteindre les tests d'acceptation il n'y avait que rarement des surprises désagréables.

Avec la sortie de la première version, nous avons eu un grand nombre de vrais utilisateurs à Östergötland (une région dans le sud de la Suède) nous fournissant un flux continu de feedback.

6. Comment nous avons structuré les équipes

Lorsque j'ai rejoint ce projet en tant que coach agile et lean en Décembre 2010, l'équipe venait juste de passer de 30 à plus de 60 personnes et elle ressentait des difficultés à s'élargir, notamment à cause des problèmes de communication et de coordination. Nous étions heureusement tous situés au même étage, chacun étant au pire à quelques dizaines de mètres des autres. Nous pouvions donc très facilement expérimenter de nouvelles manières d'organiser le projet.

Le projet s'agrandissant, nous avons changé l'organisation des équipes de la façon suivante :



Il y a maintenant cinq équipes. Une équipe d'analystes métiers, trois équipes pour le développement et une équipe pour les tests systèmes.

Les trois équipes de développement sont basées sur les fondamentaux de Scrum, c'est-à-dire que chaque équipe est colocalisée, pluridisciplinaire, auto-organisée et capable de développer et tester une fonctionnalité en entier.

L'équipe d'analystes métiers est une équipe virtuelle, dans le sens où ses membres ne sont pas colocalisés. Il y a en gros trois types d'analystes au sein de cette équipe :

- Certains analystes sont embarqués au sein des équipes de développement et accompagnent une équipe de développement jusqu'aux tests, répondant aux questions et clarifiant régulièrement les besoins.
- Quelques uns se concentrent sur la vision globale du projet et ne sont donc embarqués dans aucune des équipes de développement. Ils regardent loin dans le futur pour définir de nouvelles fonctionnalités à un très haut niveau.
- Les autres analystes sont flexibles et évoluent entre les deux rôles décrits ci-dessus selon l'importance du besoin à un moment donné.

L'équipe de tests est également une équipe virtuelle qui suit la même structure avec aussi trois types de testeurs :

- Certains testeurs sont embarqués dans une équipe de développement et aident cette équipe à tester et corriger le logiciel au niveau fonctionnel.
- Quelques uns se concentrent sur la vision globale du projet pour effectuer des tests systèmes et d'intégration haut niveau sur les versions candidates qui sortent. Celui qui coordonne ce travail est baptisé le "Général des tests du système" :o)
- Les autres testeurs sont flexibles et évoluent entre les deux rôles décrits ci-dessus selon les besoins.

Avant, les équipes étaient organisées par discipline, c'est-à-dire que nous avions des équipes distinctes d'analyses, de tests et de développement sachant que cette dernière n'embarquait ni analystes ni testeurs. Cela ne fonctionnait pas bien : plus l'équipe grossissait, plus d'importants problèmes de communication apparaissaient entre les disciplines. Il y avait une fâcheuse tendance à se consacrer à la rédaction des documents détaillés et reporter les problèmes sur les autres équipes plutôt qu'à communiquer. Chaque équipe se focalisait sur le fait de terminer ses propres tâches au lieu de se concentrer sur la construction du produit dans son entier.

Le niveau de collaboration a significativement augmenté en évoluant vers une structure du type Scrum avec des équipes pluridisciplinaires composées d'analystes, de testeurs et de développeurs colocalisés. Nous ne l'avons pas entièrement appliquée puisque nous avons conservé des analystes et des testeurs en dehors des équipes de développement pour se concentrer sur la vision globale plutôt que sur des fonctionnalités particulières. C'était parfaitement adapté à la croissance des équipes et cela nous a permis de trouver un équilibre entre vue fonctionnelle et vue système.

Etant donnée la taille du projet (plus de 60 personnes), nous avions également quelques personnes gravitant autour des équipes tenant des rôles de spécialistes et de coordinateurs. Cela incluait le chef de projet, la direction du projet, le gestionnaire de configuration, le spécialiste de la formation en ligne, les experts de tests de performance, le directeur technique, ...

7. Le cocktail quotidien

Si un jour vous passez avant 10h15 sur le plateau projet, vous aurez l'impression de tomber en plein milieu d'un cocktail ! Des gens partout, debout en petits groupes et en train de discuter.

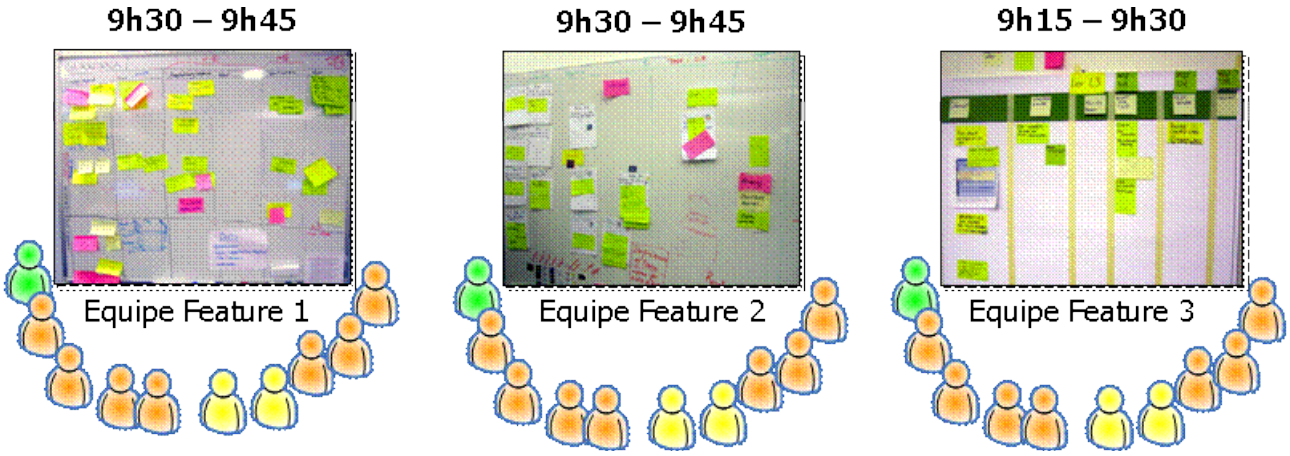
Vous y verrez des gens réunis en groupe debout en demi-cercle devant des tableaux, en pleine conversation et en train de déplacer des notes adhésives. Vous en verrez d'autres se déplacer entre les équipes. Vous assisterez à des discussions et des prises de décisions. Soudain, un groupe se sépare et une partie de ses membres va rejoindre d'autres groupes pour continuer la discussion. Parfois, de nouveaux groupes se formeront pour parler de sujets connexes.

À 10h15, le cocktail quotidien se termine et la plupart des gens retournent à leurs bureaux.

À première vue, cela peut paraître chaotique, mais c'est en fait très structuré.

1er niveau : mêlées quotidiennes des équipes de développement

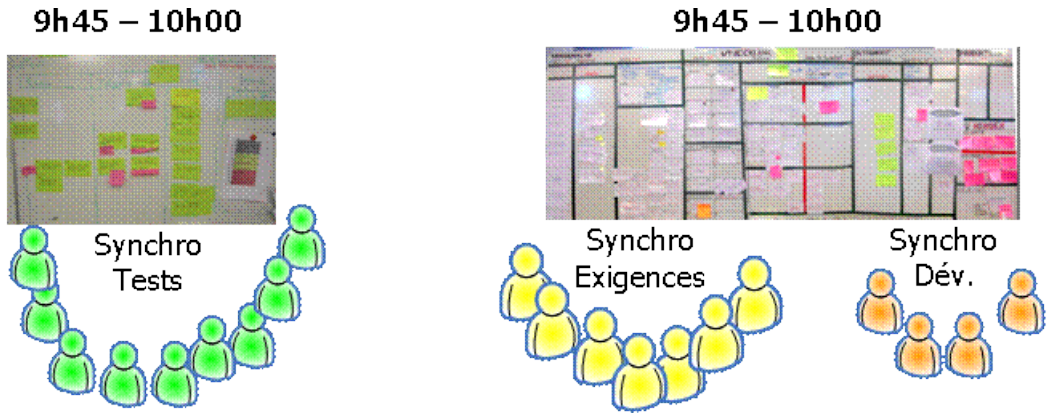
Tout commence avec les mêlées quotidiennes des équipes de développement.



Deux équipes se réunissent à 9h30 et la troisième à 9h15 (chaque équipe décide de son horaire pour se réunir). Pendant cette réunion, tous les membres de l'équipe se tiennent debout en demi-cercle face à leur tableau de tâches pour discuter du travail qui va être réalisé aujourd'hui et des problèmes et sujets à traiter. Certaines équipes utilisent les 3 questions Scrum ("Qu'est-ce que j'ai fait hier ?", "Qu'est-ce que je vais faire aujourd'hui ?", "Par quoi suis-je bloqué ?"), d'autres équipes restent plus informelles. Ces réunions durent généralement de 10 à 15 minutes et sont facilitées par un chef d'équipe (qui se rapproche assez du rôle de Scrum Master).

2ème niveau : réunions de synchronisation par type d'activité

À précisément 9h45, une deuxième série de mêlées quotidiennes a lieu : des réunions de synchronisation pour chaque type d'activité.



Tous les testeurs se regroupent face à un tableau d'avancement des tests et discutent de la manière d'utiliser au mieux leur temps dans la journée. Les testeurs embarqués viennent juste de terminer la mêlée quotidienne de leur équipe de développement, ils disposent donc de nouvelles toutes fraîches sur ce qui se passe au sein de chaque équipe.

En même temps, les analystes métiers tiennent leur réunion de synchronisation, incluant les analystes embarqués venant de terminer leur mêlée quotidienne au sein de leur équipe fonctionnelle et disposant de nouvelles toutes fraîches.

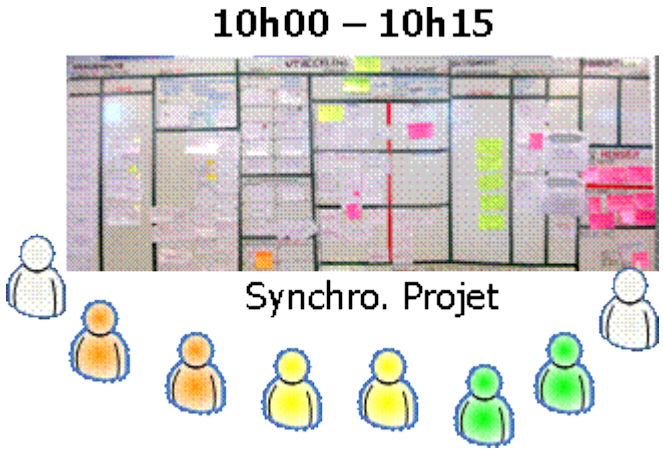
Et toujours en même temps, les chefs d'équipe de chaque équipe de développement et le directeur technique tiennent leur réunion de synchronisation sur le développement. Les chefs d'équipes de développement arrivent de leurs mêlées quotidiennes avec des nouvelles fraîches. Ils discutent de sujets techniques comme par exemple : quelle partie du code sera ajoutée aujourd'hui, quelle équipe va commencer à travailler sur quelle fonctionnalité, ainsi que des problèmes techniques et des dépendances devant être résolues.

La réunion de synchronisation des tests se tient face à un tableau reflétant l'état des tests, alors que les réunions de synchronisation des exigences fonctionnelles et des développements se tiennent face au Tableau du Projet, que je vous décrirai bientôt. Ces trois réunions se tiennent en parallèle, à quelques mètres les unes des autres, ce qui les rend un peu bruyantes et chaotiques, mais améliore le niveau de collaboration. Si quelqu'un a besoin d'une information d'une autre équipe, il n'a qu'à parcourir quelques mètres pour poser sa question dans l'autre réunion.

Certaines personnes (comme le chef de projet et moi) naviguent autour de ces réunions, en absorbant au passage ce qui se passe et en essayant de percevoir les sujets de fond qui doivent être résolus. Parfois, nous restons en dehors, d'autres fois nous rentrons dans la discussion.

3ème niveau : réunion de synchronisation du projet

Enfin, à 10h00 précises, la réunion de synchronisation du projet a lieu devant le tableau du projet.



Les personnes participant à cette réunion forment "l'équipe transverse" ("tvärgrupp" en suédois), ce qui signifie simplement qu'il s'agit d'une personne de chaque type d'activité et d'une personne de chaque équipe de développement, en y ajoutant le chef de projet et le responsable de la gestion de configuration. Une belle coupe transversale de l'ensemble du projet.

La réunion de synchronisation du projet est celle où l'on s'intéresse à la vision d'ensemble en se concentrant sur le flux de fonctionnalités, de l'analyse à la production et sur ce que chaque équipe fait aujourd'hui : Qu'est-ce-qui peut bloquer le flux en ce

moment ? Où se situe le goulet d'étranglement et comment l'atténuer ? Où apparaîtra le prochain goulet d'étranglement ? Sommes nous sur la bonne voie pour respecter le planning de la version ? Y a-t-il quelqu'un qui ne sait pas ce qu'il a à faire aujourd'hui ?

Voilà. Un total de 7 mêlées quotidiennes par jour, organisées sur trois niveaux. Chaque réunion est limitée à 15 minutes, chaque réunion a un noyau dur de participants présents quotidiennement, et chaque réunion est publique de telle façon que n'importe qui peut s'inviter à une réunion pour savoir ce qui s'y passe ou même contribuer. Et tout ça avant 10h15.

Si certains points importants sont soulevés pendant une mêlée quotidienne et ne peuvent pas être résolus en moins de 15 minutes, nous planifions une réunion de suivi avec les personnes nécessaires à la résolution du problème. Certaines des discussions les plus intéressantes et apportant le plus de valeur, ont lieu juste après la réunion de synchronisation du projet, les gens se réunissent en petits groupes pour traiter des sujets ayant émergé pendant les mêlées quotidiennes.

Cette structure de mêlées quotidiennes s'est construite au fur et à mesure, comme tout le reste. Lorsque nous avons commencé à avoir un "cocktail" (au fait, c'est moi qui emploie cette expression, il ne s'agit pas de l'expression officiellement utilisée pendant le projet...), j'étais préoccupé par le fait que les gens pensaient qu'il y avait trop de réunions. Ce ne fut pas le cas, les membres des différentes équipes ont bien insisté sur le fait que ces réunions étaient très importantes, et j'ai observé qu'il y avait beaucoup d'énergie et que les problèmes étaient résolus.

La plupart des gens n'ont besoin de participer qu'à une seule réunion. Seul un petit nombre doivent se rendre à deux réunions. Le chef d'une équipe de développement participe aussi bien à sa mêlée qu'à la réunion de synchronisation des développements. Le testeur embarqué dans une équipe de développement participe à la mêlée ainsi qu'à la réunion de synchronisation des tests, etc. C'est un moyen très efficace de "relier" les canaux de communication et de s'assurer que les informations importantes et les décisions se propagent rapidement à travers tout le projet.

Il est apparu que de nombreux problèmes, qui auraient classiquement donné lieu à la création de documents et de règles, ont été résolus directement lors de ces réunions du matin. Un exemple concret concerne la décision de savoir quelle équipe développe quelle fonctionnalité, ou s'il faut consacrer son temps à développer une fonctionnalité visible du client plutôt qu'une amélioration de l'infrastructure technique invisible au client. Au lieu de mettre en oeuvre des règles pour ça, les équipes en ont simplement parlé lors des mêlées quotidiennes et pris les décisions à la volée basées sur la situation actuelle. C'est la clé pour rester agile au sein d'un gros projet et ne pas s'enliser dans la bureaucratie.

8. Le tableau de bord du projet

Le tableau de bord du projet est le centre de communication du projet.

Si vous pratiquez Kanban, vous reconnaîtrez un système kanban, ce qui signifie que nous suivons le flux de valeur de l'idée à la production, et que nous limitons la quantité de travail en cours à chaque étape du processus.

Voici un résumé de la signification de chaque colonne :

La colonne à l'extrême gauche est celle où arrivent les idées, des domaines fonctionnels de haut niveau (« épopées ») écrites sur des fiches cartonnées. La carte est tirée dans la deuxième colonne (« analyse en cours »), où elle est analysée puis découpée en fonctionnalités, qui sont collectées dans la troisième colonne. Ainsi la troisième colonne correspond au Product Backlog de Scrum. La plupart des cartes de fonctionnalités sont écrites avec le format classique d'une histoire utilisateur « En tant que X, je veux Y, afin de Z ».

Les 10 premières fonctionnalités sont sélectionnées et tirées dans la colonne des « 10 fonctionnalités suivantes ». Cela est généralement pratiqué lors d'une réunion, toutes les deux semaines, qui correspond grosso modo à la réunion de planification de sprint de Scrum (d'ailleurs nous l'appelons ainsi).

Les 3 équipes de développement tirent continuellement les cartes à partir de la colonne des « 10 fonctionnalités suivantes » dans leur propre colonne « dév en cours » lorsqu'ils ont la capacité suffisante. lorsque la fonctionnalité est développée et testée au niveau fonctionnalité la carte est placée dans la colonne « Prêtes pour les tests du système ».

De manière régulière, l'équipe de test utilisera la colonne « Prêtes pour les tests du système » et tirera toutes ces cartes dans la colonne « Test du système en cours » (elle créera une branche correspondante dans le gestionnaire de configuration). Une fois les tests du système réalisés, les membres de l'équipe livreront un environnement de tests d'acceptation, déplaceront les cartes dans la colonne « prêtes pour les tests d'acceptation », et commenceront un nouveau tour de tests du système sur toute fonctionnalité qui aura été réalisée entre temps. Ceci est un grand changement culturel – du « gros test du système à la fin du cycle de livraison » au « test du système en continu » (avec quelques traitements différés).

Environ tous les deux mois, quelques vrais utilisateurs viennent passer quelques jours pour faire des tests d'acceptation (en essayant simplement le système et en donnant du feedback), alors nous déplaçons les cartes dans la colonne « test d'acceptation en cours ». Lorsqu'ils finissent les tests et que les dernières anomalies sont trouvées et corrigées, les cartes sont déplacées dans « Prêtes pour la production » et peu après (une fois le système livré), elles sont déplacées dans la colonne « en production ». Les cartes restent là quelques semaines (afin que nous puissions fêter leur mise en production), puis sont enlevées pour faire de la place lorsque de nouvelles cartes arrivent.

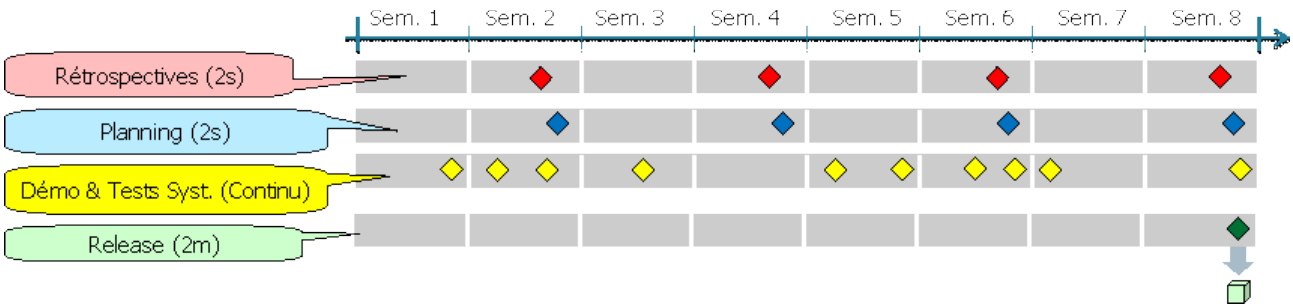
L'observateur occasionnel, jetant à un coup d'oeil à ce tableau, pourrait croire qu'il s'agit d'un processus en cascade : analyse des exigences => développement => test

système => test d'acceptation => production. Néanmoins, il y a une grosse différence : dans un modèle en cascade, les exigences sont exhaustives avant que le développement ne commence, et le développement est achevé avant que les tests ne débutent. Dans un système Kanban, ces phases se déroulent en parallèle. Pendant qu'un ensemble de fonctionnalités est en cours de test d'acceptation par des utilisateurs, un autre ensemble de fonctionnalités est en cours de test système, alors qu'un troisième ensemble de fonctionnalités en cours de développement et que le quatrième ensemble est en cours d'analyse et de découpage en histoires d'utilisateur. C'est un flux continu de valeur de l'idée à la production.

En fait, je devrais dire semi-continu. Dans notre cas, c'est un flux de valeur plus ou moins continu de « l'idée au test d'acceptation ». De nouvelles fonctionnalités sont livrées en production environ un mois sur deux et en test d'acceptation en liaison avec ça, du coup les fonctionnalités sont « prêtes pour les tests d'acceptation » pendant quelques semaines. Bien que j'espère que nous pourrons améliorer ceci dans le futur, cela ne s'est pas révélé être vraiment un problème. Comme nous avons des utilisateurs sur site nous donnant leur feedback pendant le développement, il se trouve que lorsqu'une fonctionnalité atteint la colonne « prêtes pour les tests d'acceptation », elle fonctionne à peu près comme prévu. Il est rare de trouver un problème sérieux après cette étape.

9. Nos cadences

Voici un aperçu de nos cadences :



- les rétrospectives se tiennent toutes les 2 semaines (certaines équipes les font toutes les semaines)
- la planification a lieu toutes les 2 semaines (approximativement)
- la démo et le test système sont faits de façon continue, au fur et à mesure que les fonctionnalités sont finies
- la mise en production est faite environ tous les 2 mois

Nous avons évolué progressivement vers un modèle « à la Scrum ». Au début les rétrospectives étaient deux fois plus fréquentes que les réunions de planification, maintenant elles ont lieu toutes les deux semaines, l'une après l'autre à un jour d'intervalle. Les démos et revues sont faites continuellement mais nous envisageons maintenant de faire une démo/revue du produit à haut niveau une semaine sur deux. Et devinez quoi – faire des rétrospectives, des planifications et des démos ensemble à la même cadence est la base de la définition d'un sprint en Scrum :o)
Cette évolution vers un modèle de plus en plus « à la Scrum » n'était pas vraiment intentionnelle, c'est juste le résultat d'une série d'améliorations de processus, déclenchés par les problèmes réels de tous les jours.
Cela semble être courant, je vois de nombreuses équipes Kanban qui découvrent progressivement (ou parfois redécouvrent) la valeur de nombreuses pratiques Scrum. En fait, parfois les équipes Kanban commencent à faire du Kanban parce qu'elles n'aiment pas Scrum et découvrent plus tard que Scrum n'est finalement pas si mal et que leurs problèmes ont été *révélés* et non pas *causés* par Scrum. Leur problème *réel* était qu'elles avaient fait du Scrum trop « au pied de la lettre » au lieu d'inspecter et de l'adapter à leur contexte.

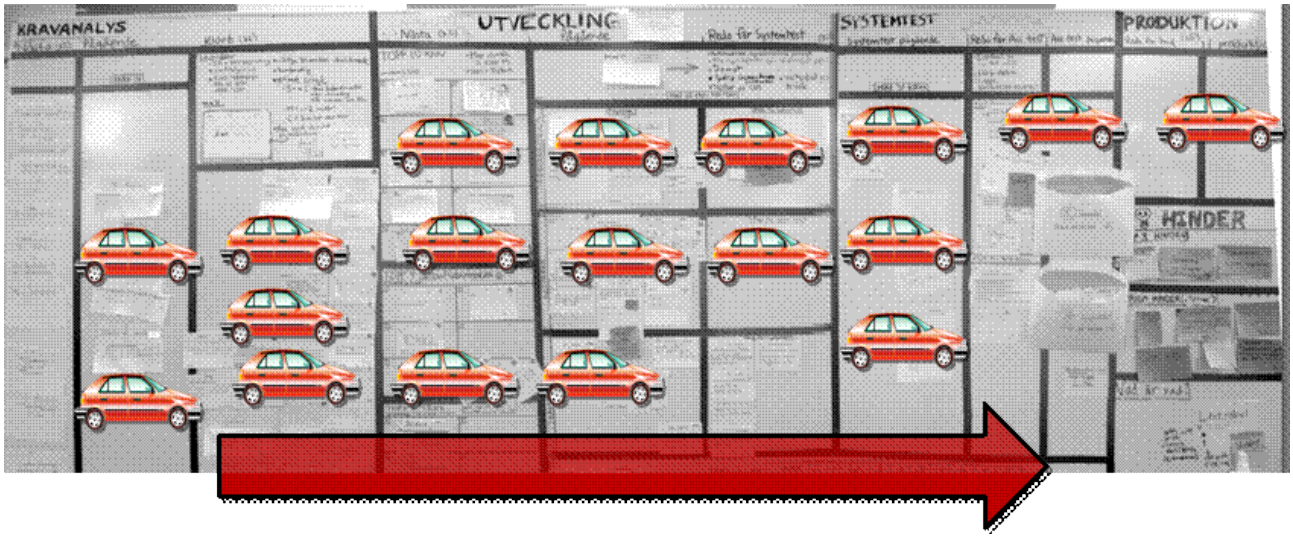
Plus sur ce sujet dans mon autre livre « Kanban & Scrum, tirer le meilleur des 2 » (NdT : traduit en français).

Maintenant, revenons à nos moutons.

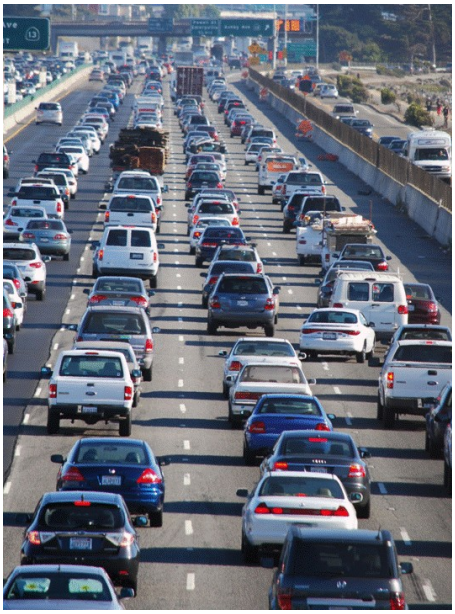
10. Comment nous avons traité les problèmes urgents et les obstacles

Il a été très utile d'utiliser une métaphore, celle du trafic routier, quand il a été question de regarder le tableau du projet.

Pensez donc au tableau comme un réseau routier, avec chaque carte représentant une voiture essayant de traverser le tableau de la gauche vers la droite.

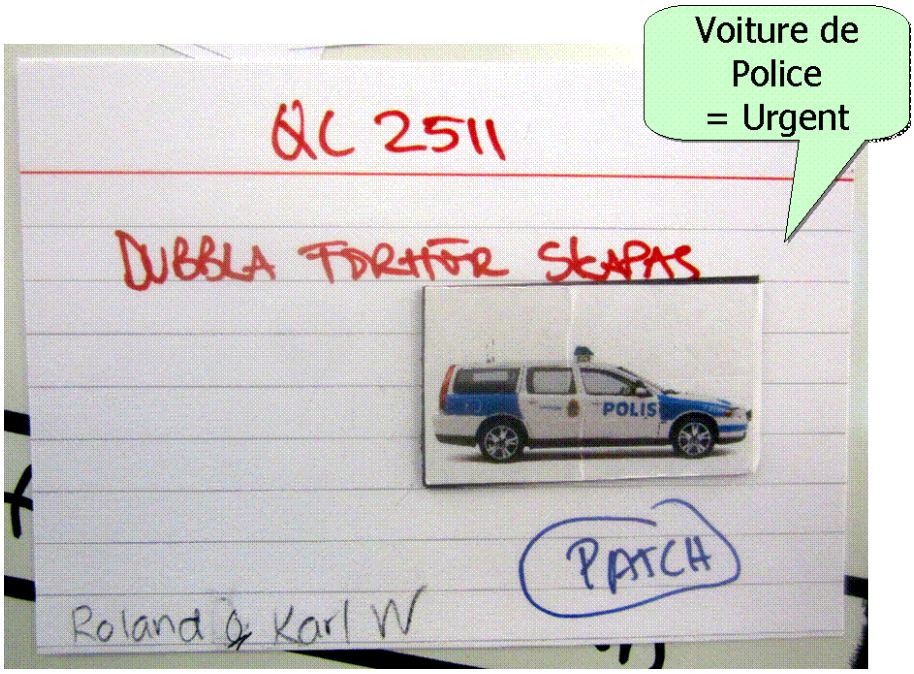


Nous souhaitons optimiser le flux, cependant nous NE voulons PAS remplir le tableau. Nous savons tous ce qu'il arrive quand le flux de circulation est plein à 100% : le trafic ralentit jusqu'à s'arrêter.

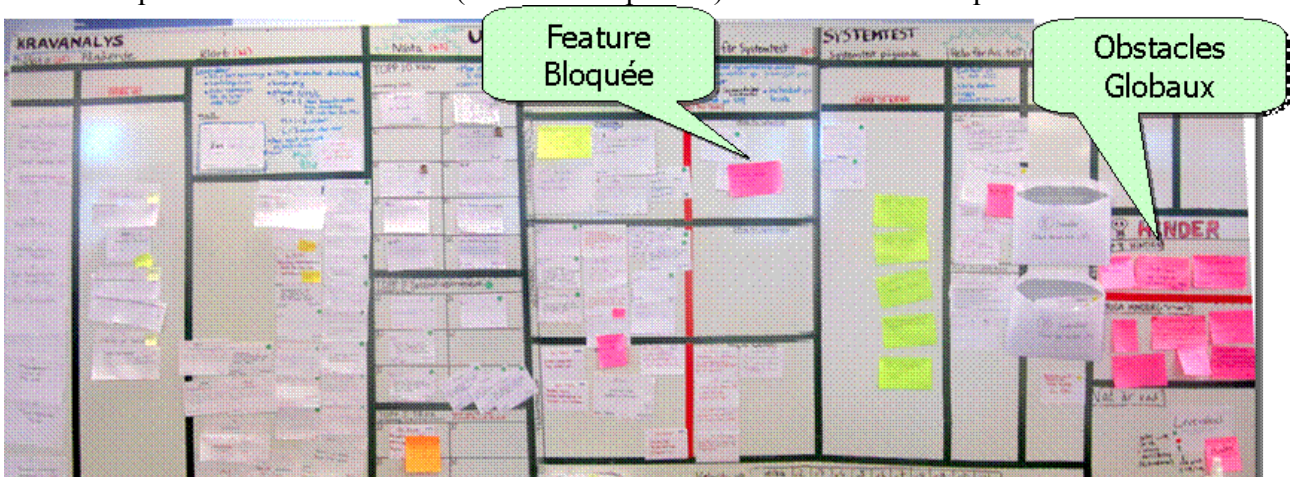


Nous avons besoin d'espace, ou de « mou », pour absorber les variations et permettre au flux de s'écouler vite. Du mou dans le système, non seulement cela donne un flux plus rapide, mais cela facilite l'escalade des problèmes, en vue de leur résolution. Pour rester sur la métaphore, nous

utilisons des aimants représentant voiture de police pour indiquer des éléments urgents et nécessitant un traitement spécial pour traverser le système plus rapidement.



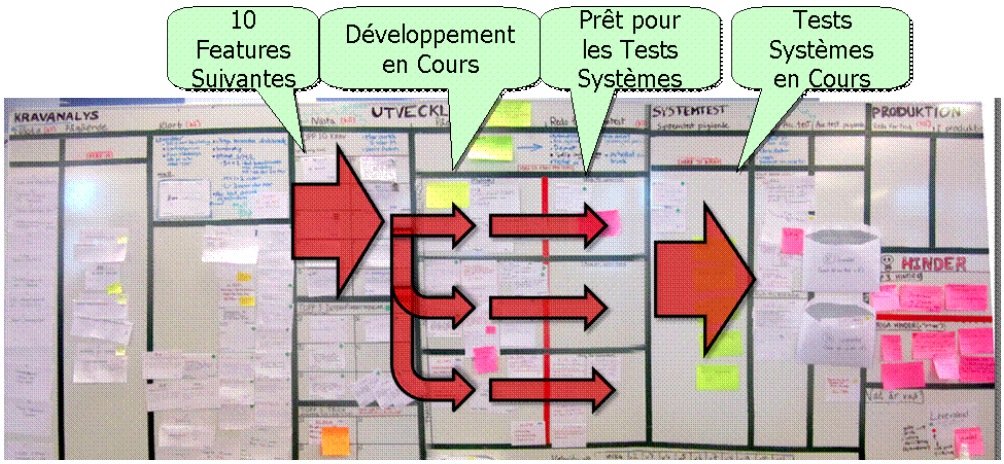
Nous marquons aussi les obstacles (« routes bloquées ») en utilisant des étiquettes roses.



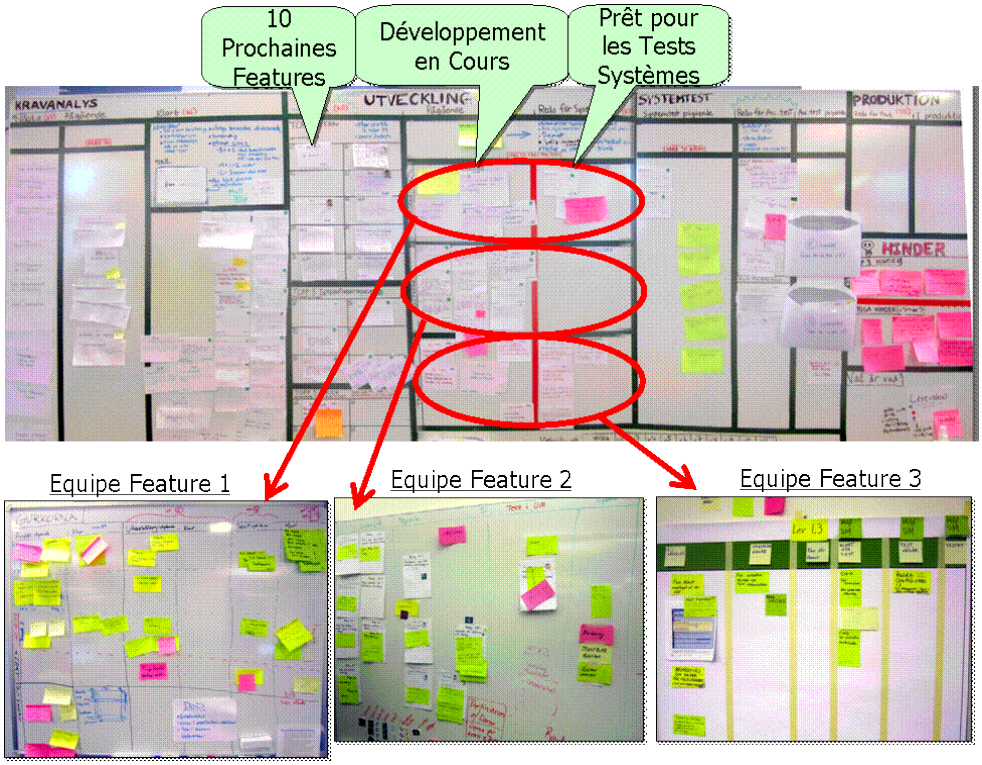
Si une fonctionnalité est bloquée (par exemple parce que nous n'avons pas l'accès à un sous-système tierce nécessaire au test de cette fonctionnalité), alors nous mettons une étiquette rose sur cette fonctionnalité. Sur cette étiquette, nous notons une description du problème et la date à laquelle il est arrivé. Il y a aussi une section sur le côté droit, pour les « 3 obstacles majeurs ». Elle sert à des problèmes plus généraux qui ne sont pas liés à une fonctionnalité particulière (par exemple un environnement de build qui ne marche pas).
Pendant les réunions quotidiennes, nous nous concentrons sur l'élimination de ces obstacles. C'est comme pour le trafic routier, un obstacle qui reste trop longtemps va provoquer des perturbations dans tout le système. Et comme la vitesse du flux est bornée par celle dans le goulet d'étranglement, nous devons concentrer tous nos efforts pour résoudre ces blocages.

11. Comment nous avons mis en place les tableaux Kanban

Sur le tableau du projet, les colonnes concernant le développement ont été découpées en trois couloirs horizontaux, un pour chaque équipe de développement. Chaque fonctionnalité circule de la colonne "10 prochaines fonctionnalités" vers l'une des trois colonnes représentant une équipe de développement. Lorsque cette équipe a développé la fonctionnalité et l'a testée, elle est alors "Prête pour les tests du système". Lorsque l'équipe des tests du système a terminé de traiter une série de fonctionnalités, elle tire toutes les cartes de chaque colonne "Prête pour les test du système" de chaque équipe de developpement pour les placer dans la colonne "Tests du système en cours".



Chaque fois qu'une équipe tire une fonctionnalité de la colonne "10 prochaines fonctionnalités" vers la colonne "Développement en cours", elle clone la carte correspondant à la fonctionnalité et la place sur son propre tableau.



L'équipe décompose ensuite le travail en tâches et l'inscrit sur des notes adhésives liés à la fonctionnalité. Cette étape est typiquement réalisée lors d'une "réunion d'analyse" où les analystes métiers, les testeurs et les développeurs collaborent pour esquisser la conception de la fonctionnalité concernée et identifier les tâches essentielles à sa réalisation. Chaque tâche se résume normalement à un verbe d'action concret tel que "écrire le code IHM" ou "créer les tables de la BDD" ou encore "concevoir le protocole".

La plupart des équipes disposent également de leurs avatars sur aimants pour indiquer qui travaille sur quelle tâche. Votre avatar en dit long sur votre personnalité :o)

Le tableau du projet contient donc des cartes de fonctionnalité, et chaque équipe de développement dispose de son propre tableau avec les fonctionnalités sur lesquelles elle travaille ainsi que le découpage en tâches associées. Imaginez que vous "double-cliquez" sur une fonctionnalité du tableau du projet et que vous zoomiez sur la fonctionnalié du tableau de l'équipe concernée, vous y verrez les tâches associées à cette fonctionnalité et qui travaille sur quelle tâche.

Comme vous pouvez le voir sur les photos ci-dessus, chaque équipe a sa propre façon d'agencer son tableau. Nous n'avons pas essayé de le standardiser, au contraire, nous avons préféré les laisser à chaque fois trouver la structure de tableau leur convenant le mieux. Même si la plupart des équipes utilisent sur leurs tableaux les limites de TAF, la Définition du Fini et les avatars sur aimants.

Ce système à 2 niveaux de tableaux physiques Kanban a effectivement bien marché, même s'il y a eu un peu de confusion au début pour trouver comment les maintenir synchronisés. Il est clair que les tableaux sont devenus le centre de gravité du projet, les équipes se regroupant naturellement autour de leurs tableaux à chaque fois qu'elles ont besoin de synchroniser leur travail ou de résoudre leurs problèmes. La plupart des équipiers se concentrent sur leurs tableaux alors que les chefs d'équipe et les managers se concentrent sur les deux niveaux, que ce soit le tableau de l'équipe ou le tableau du projet.

Au fur et à mesure du temps, de plus en plus d'équipiers ont commencé à s'intéresser au tableau du projet, ce qui est un bon indicateur de l'intérêt croissant des personnes à disposer de la vision d'ensemble au-delà de leur propre travail.

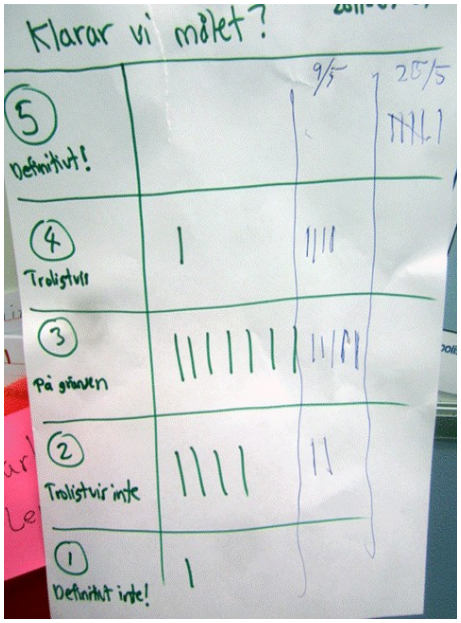
12. Suivi de l'atteinte de l'objectif majeur du projet

L'objectif principal du projet est généralement affiché sur le tableau Kanban. Par exemple, pendant le 1er trimestre 2011, nous avions l'objectif de "Livrer le 5 avril une version sans anomalie bloquante sur tout le territoire", et un jalon sur cette période consistait à livrer deux comtés le 14 mars. Sur les autres périodes, nous avions des objectifs différents.

Environ une fois par semaine, nous menions un test de confiance. Typiquement, le chef de projet demandait lors de la réunion de synchronisation du projet "Pensez-vous que nous allons atteindre l'objectif ?" et chacun inscrivait un chiffre entre 1 et 5 (parfois nous utilisions juste nos doigts).

- 5 = Absolument !
- 4 = Probablement
- 3 = De justesse
- 2 = Probablement pas
- 1 = N'y comptez pas !

Voici un exemple :



Ce test de confiance était réalisé chaque semaine, et cette feuille montre trois tours de vote. La première semaine (colonne la plus à gauche), il y avait peu de confiance dans l'objectif, la semaine suivante la confiance était montée et la semaine d'après il n'y avait que des cinq !

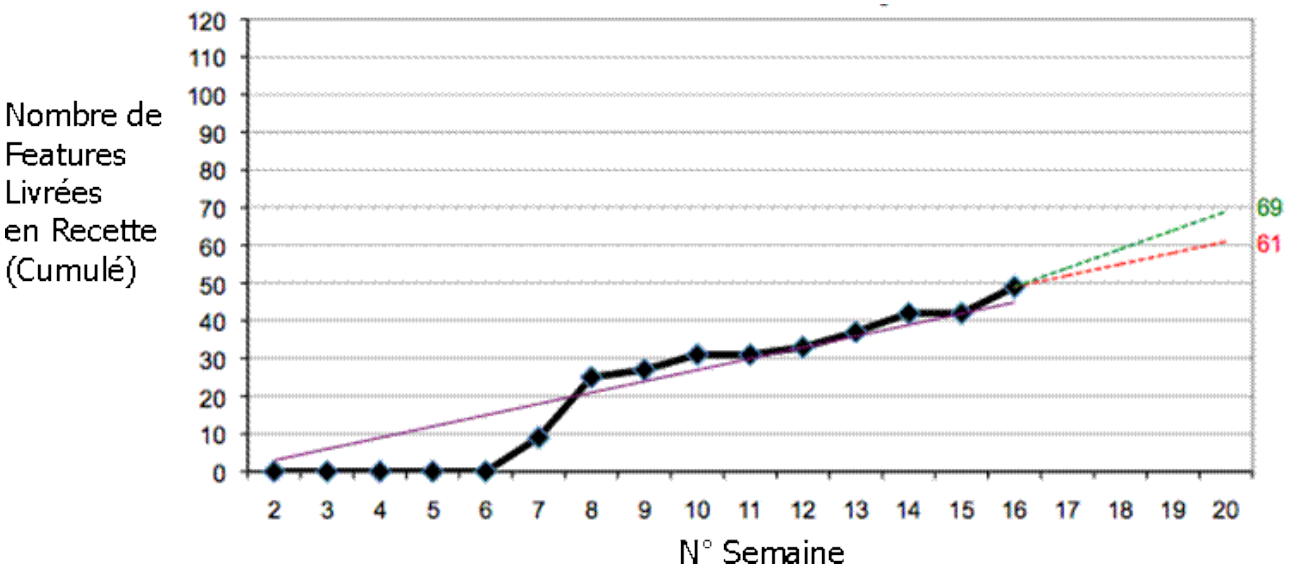
Quand nous commençions à voir des 2 et des 1, nous réévaluions l'objectif du projet et discutions de ce qu'il fallait changer pour améliorer notre confiance. Typiquement, l'une des choses suivantes :

- Supprimer un obstacle
- Atténuer un goulet d'étranglement

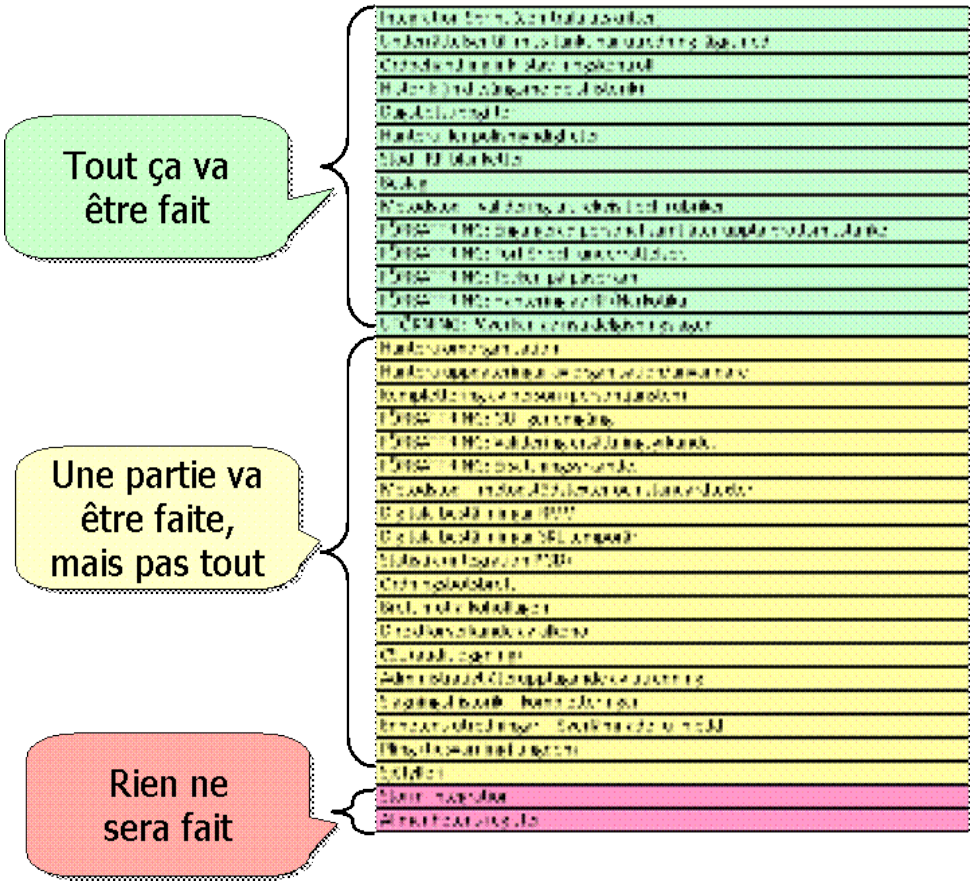
- Réduire le périmètre
- Ajuster l'objectif
- Travailler plus dur

N'importe lequel des quatre premiers choix est préférable au dernier, puisque la cause racine du problème n'est généralement *pas* que les gens ne travaillent pas assez dur. En fait, parfois la cause racine du problème était que nous travaillions *trop* dur, et que nous ne prenions pas le temps de réfléchir.

Les votes sont essentiellement basés sur l'instinct, mais aussi, et jusqu'à un certain point, sur les informations visibles comme les cartes sur le tableau, des métriques comme le temps de cycle et la vélocité (décrits au chapitre 19), et des graphiques comme le burnup de fonctionnalités :



En utilisant les tendances optimiste ou pessimiste, nous pouvons faire des projections quantitatives sur le nombre de fonctionnalités qui seront terminées d'ici la date de livraison :

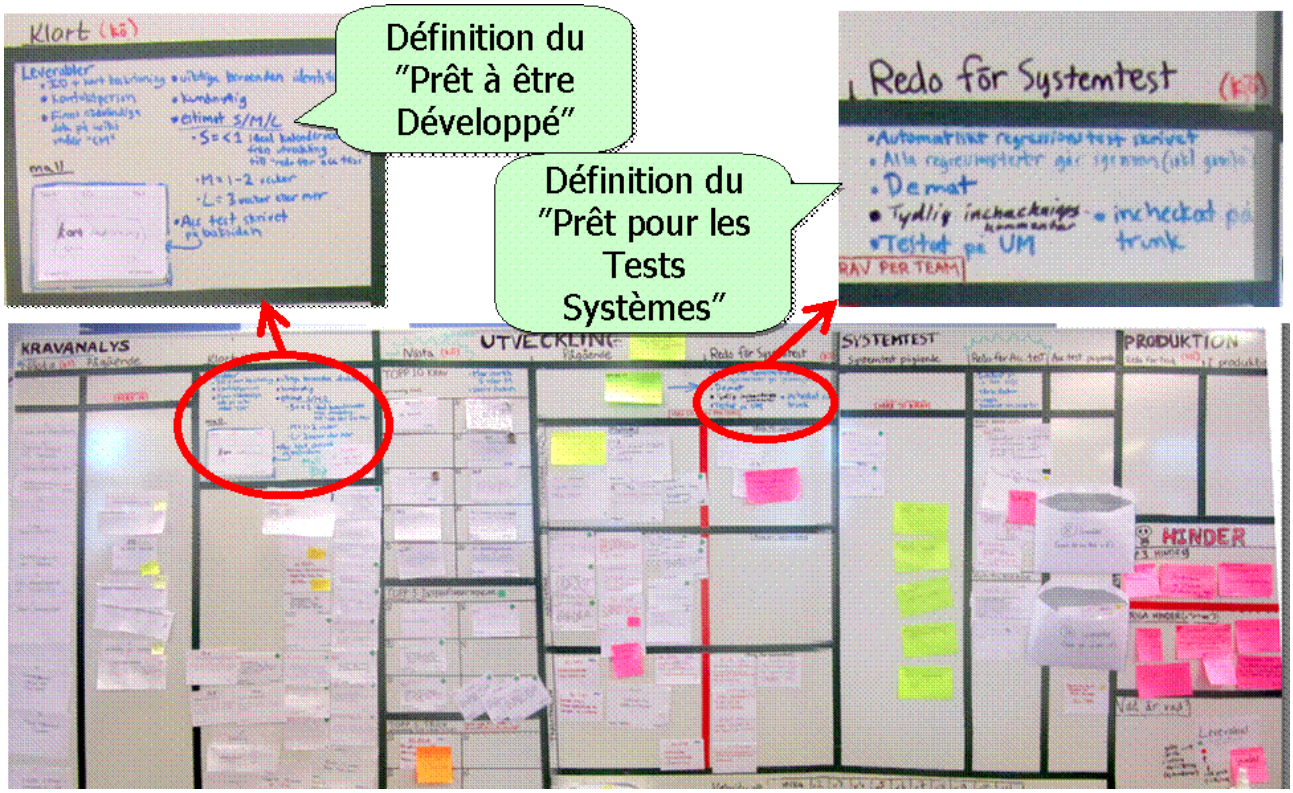


Ce genre de test de confiance en continu est une technique très simple et très utile pour détecter et éviter les longues marches vers une mort inexorable (c'est-à-dire les projets où chacun sait que cela va échouer, mais continue quand même à avancer consciencieusement). Si les gens peuvent se mettre d'accord sur un objectif auquel ils croient, cela a un immense effet positif sur l'auto-organisation et la collaboration. Inversement, si les gens ne comprennent pas l'objectif, ou qu'ils ne croient pas en l'atteinte de l'objectif, ils vont inconsciemment se désolidariser de l'objectif et se concentrer sur d'autres objectifs comme par exemple "prendre plaisir à coder" ou "juste réaliser mon travail et rentrer à la maison".

Je recommande fortement d'avoir un objectif clair ainsi qu'un test de confiance périodique, quelle que soit la nature du projet dans lequel vous êtes impliqué. Le coût en est très faible, et le bénéfice immense.

13. Comment nous avons défini le Prêt et le Fini

Le texte écrit en bleu, au-dessus de la plupart des colonnes du tableau du projet, représente la "définition du fini" de la colonne concernée (et représente donc également la "définition du prêt" pour la colonne suivante). Les deux plus importantes définitions sont la "Définition du Prêt pour le développement" et la "Définition du Prêt pour les Tests du Système", car c'est là où nous rencontrons le plus de problèmes.



Définition du "Prêt pour le Développement"

La colonne "Prêt pour le Développement" signifie principalement qu'il y a "ici un paquet de fonctionnalités que nous avons décomposées, estimées et clarifiées, mais dont nous n'avons pas encore décidé lesquelles nous allions développer et dans quel ordre." Cela correspond donc en gros au Backlog Produit de Scrum.

Pour qu'une fonctionnalité soit prête à être développée, elle doit :

- avoir un **ID**. Cet identifiant est utilisé comme une clé lorsque nous recherchons des informations plus détaillées à propos de cette fonctionnalité, dans le cas où des spécifications de cas d'utilisation ou d'autres documents existeraient.
- avoir le **nom d'un contact**. Typiquement le nom de l'analyste métier qui connaît le mieux cette fonctionnalité.
- apporter de la **valeur** aux clients. Lorsque nous décomposons des épopées en histoires livrables, nous devons nous assurer que nous n'avons pas au passage

perdu la valeur métier. Les analystes métiers ont le dernier mot sur ce point précis.
- avoir été **estimée** par l'équipe. Nous utilisons les tailles de t-shirts (Petit, Moyen, Grand). Les estimations sont normalement effectuées par un petit groupe composé d'un testeur, d'un développeur et d'un analyste métier qui jouent au planning poker. Chacune des tailles de t-shirts fournit une indication approximative :
    - **Petit** signifie "dans un contexte idéal, l'équipe passera moins d'une semaine pour que la fonctionnalité soit Prête pour les Tests d'Acceptation".
    - **Moyen** signifie 1 à 2 semaines (dans un contexte idéal).
    - **Grand** signifie plus de 2 semaines.
- disposer d'un scénario de tests d'acceptation écrit au dos de la carte. Il s'agit d'un ensemble concret d'étapes décrivant un scénario de tests classique, par exemple :
    - "Le policier Joe se connecte, recherche le Dossier #235 et le clôt. Il recherche ensuite à nouveau le Dossier #235 et voit qu'il est clos."

Définition de "Prêt pour les Tests du Système"

"Prêt pour les Tests du Système" signifie que l'équipe de développement a fait tout ce qu'elle pouvait pour garantir que cette fonctionnalité marche sans anomalie majeure. L'équipe s'est par contre concentrée sur le fait de tester uniquement la fonctionnalité, et non la version entière dont elle fait partie.

Les tests du système constituent depuis longtemps un goulet d'étranglement. L'une des raisons majeures est le nombre important d'anomalies inutiles détectées lors de ces tests. Par "anomalies inutiles", je veux dire des anomalies de la fonctionnalité qui auraient pu être détectées bien avant les tests du système. Notre "définition du Prêt pour les Tests du Système" est donc là pour maintenir le niveau de qualité à un haut niveau et détecter ces anomalies gênantes plus tôt. Elle sert également à responsabiliser l'équipe de développement sur la qualité et lui permettre de consacrer le temps nécessaire à s'assurer que la fonctionnalité marche réellement, avant de la livrer pour les tests du système et de passer à la suite.

Notre définition de "Prêt pour les Tests du Système" est donc :

- **Tests d'acceptation automatisés.** Cela signifie qu'un ensemble de tests d'intégration ou d'acceptation de bout en bout ont été automatisés. Nous avons pour cela utilisé Selenium (qui exécute directement les tests sur l'IHM web) avant de passer sur Concordion. Les tests Selenium étaient trop fragiles et Concordion est très bien adapté avec notre souhait d'utilisation des Spécifications par l'Exemple (pour plus d'infos sur ce sujet, je vous renvoie à www.specificationbyexample.com).
- **Tests de non-régression passés avec succès.** Cela signifie que tous les tests automatisés des fonctionnalités précédemment réalisées passent avec succès.
- **Démonstration.** Cela signifie que l'équipe a exécuté une démonstration informelle de la fonctionnalité à quelqu'un, par exemple à un client sur site ou le référent de cette fonctionnalité.
- **Commentaires clairs lors du check-in.** Lorsque vous fournissez le code concernant la fonctionnalité, les commentaires doivent être labellisés avec l'ID de

la fonctionnalité ainsi qu'une explication aisément compréhensible de ce qui a été réalisé.

- **Tests dans l'environnement de développement.** Chaque équipe dispose d'un environnement de tests qui lui est dédié, et la fonctionnalité doit y être testée (et non pas juste sur "ma machine").
- **Merge dans la branche principale.** Le code de cette fonctionnalité doit être fusionné à la branche principale et les conflits doivent être résolus.

Amélioration du niveau de collaboration

Ces deux déclarations de "Définition du Prêt pour le Développement" et de "Définition du Prêt pour les Tests du Système" ont considérablement amélioré le niveau de collaboration entre les spécifications, les tests et le développement. Lorsque j'ai fait un sondage rapide sur les changements apportés au processus jusque là, l'amélioration la plus fréquemment mentionnée portait sur le niveau de collaboration entre les disciplines.

Auparavant, chaque discipline se concentrait essentiellement sur "sa" partie du tableau du projet. Les analystes métiers regardaient uniquement la partie gauche du tableau du projet, et considéraient leur travail "fini" sur une fonctionnalité une fois qu'un document de spécifications avait été rédigé. Les développeurs regardaient uniquement le milieu du tableau et les testeurs uniquement la partie droite. Les testeurs n'étaient pas impliqués lors de la rédaction des spécifications, donc une fois que la fonctionnalité avait atteint la phase de tests, il y régnait toujours une certaine confusion sur la façon dont elle était supposée marcher, et beaucoup d'effort était dépensé à discuter sur le niveau de détail nécessaire à apporter dans les documents de spécifications.

Ces problèmes ont disparu petit à petit (ou ont été, tout au moins, considérablement réduits) en l'espace de quelques semaines une fois que chacun se fut approprié les définitions. La "définition du Prêt pour le Développement" peut être satisfaite seulement si les disciplines travaillent ensemble pour estimer les fonctionnalités, les décomposer en livrables suffisamment petits sans pour cela trop perdre en valeur métier, et se mettre d'accord sur les tests d'acceptation. C'est ce niveau d'exigence qui a conduit à ce niveau de collaboration.

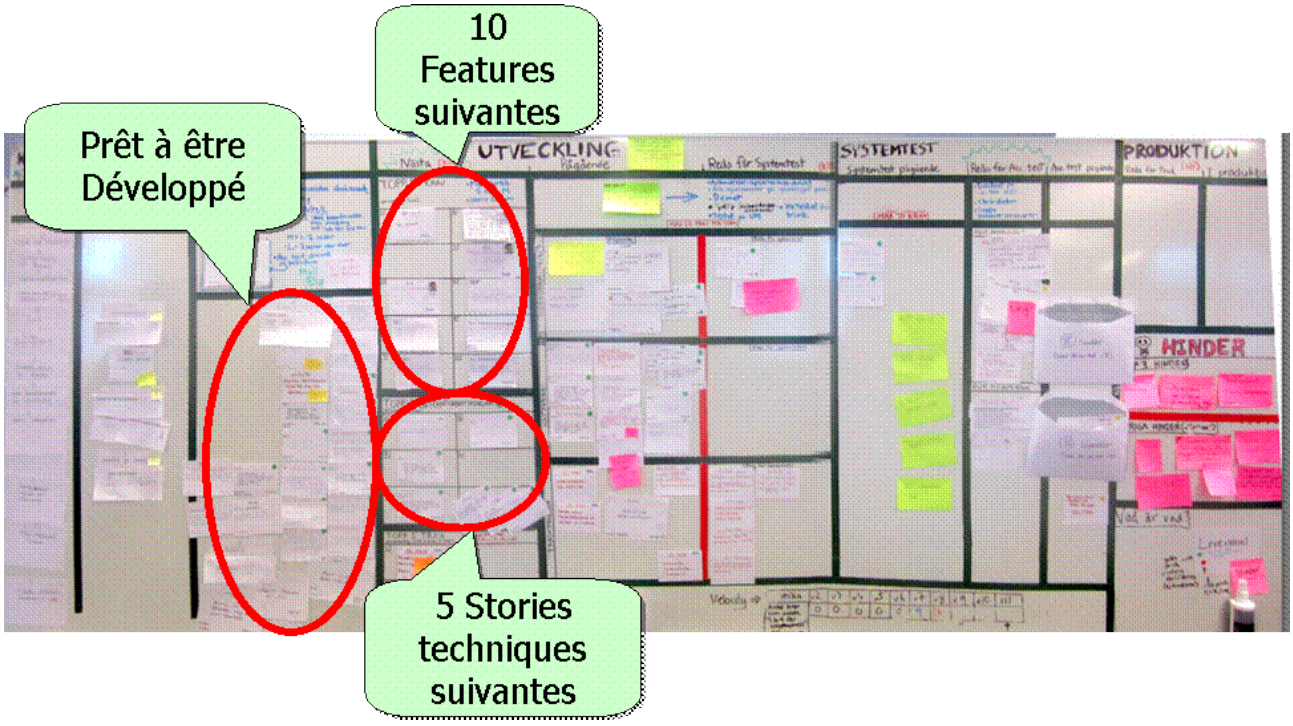
De la même manière, la "définition du Prêt pour les Tests du Système" peut être uniquement satisfaite si toutes les disciplines travaillent ensemble pour exécuter les tests au niveau de la fonctionnalité (à la fois les tests automatisés et les tests exploratoires manuels), et pour déterminer si cette fonctionnalité est suffisamment valide pour être livrée.

Ce besoin de collaboration continue a conduit les équipes de tests et de spécifications à se mettre d'accord pour "prêter" certains de leurs membres à chacune des équipes de développement.

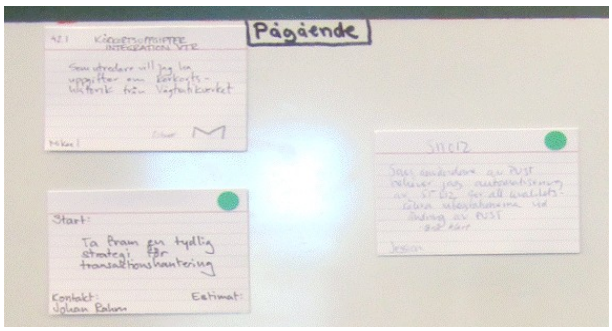
14. Comment nous avons géré les histoires techniques

Les histoires techniques sont des choses qui doivent être faites, mais qui n'ont aucun intérêt pour le client : mettre à jour la version de la base de données, enlever du code obsolète, remanier une conception désordonnée, ou encore rattraper du retard sur la mise en place de tests automatisés pour d'anciennes fonctionnalités, etc. En fait, nous appelons ceci des « améliorations internes » mais avec le recul « histoires techniques » est probablement un meilleur terme, étant donné que nous parlons de choses devant être faites pour le produit, et non d'améliorations de processus.

La section « 5 prochaines histoires techniques » figurant juste en dessous de « 10 prochaines fonctionnalités », constitue avec cette dernière deux files parallèles d'approvisionnement pour le développement. Toutes les autres histoires techniques et fonctionnalités sont empilées dans la colonne « Prêtes pour le développement » et non priorisées. Nous gagnons beaucoup de temps en sélectionnant continuellement les 10 prochaines fonctionnalités et 5 histoires techniques suivantes, plutôt que d'essayer d'ordonner tout le backlog.



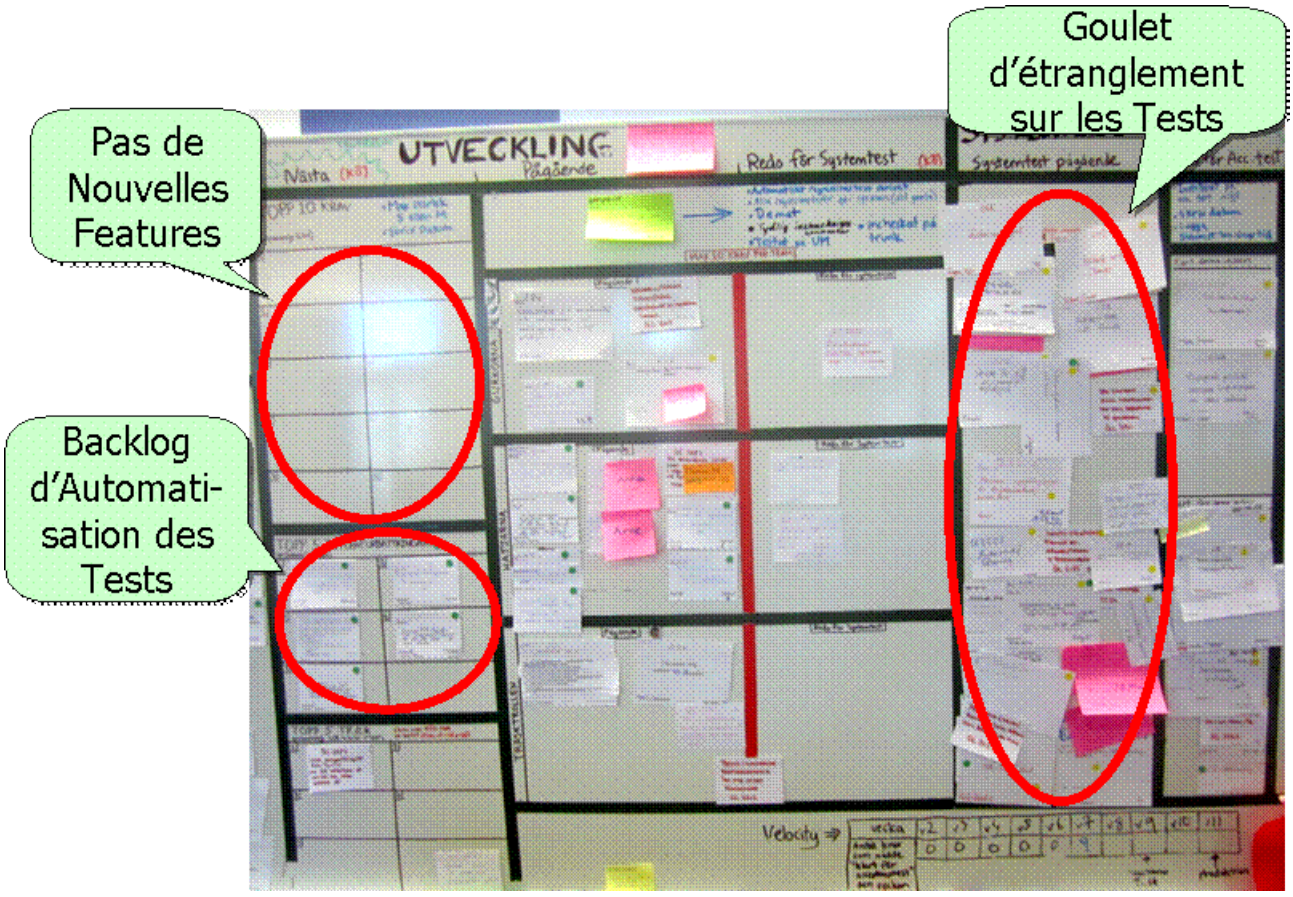
Lorsqu'une équipe de développement a la capacité de commencer quelque chose de nouveau, soit elle tire une fonctionnalité des « 10 prochaines fonctionnalités » soit elle tire une carte technique des « 5 prochaines histoires techniques ». Les histoires techniques sont distinguées des autres par un point vert dans un coin de la carte, le tableau du projet révèle donc comment nous équilibrons notre temps entre les fonctionnalités et les histoires techniques.



Nous n'avons pas de règles figées pour définir un équilibre « correct », à la place nous discutons continuellement et ajustons l'équilibre au cours des mêlées quotidiennes, principalement pendant celles de synchronisation de projets et de développement.

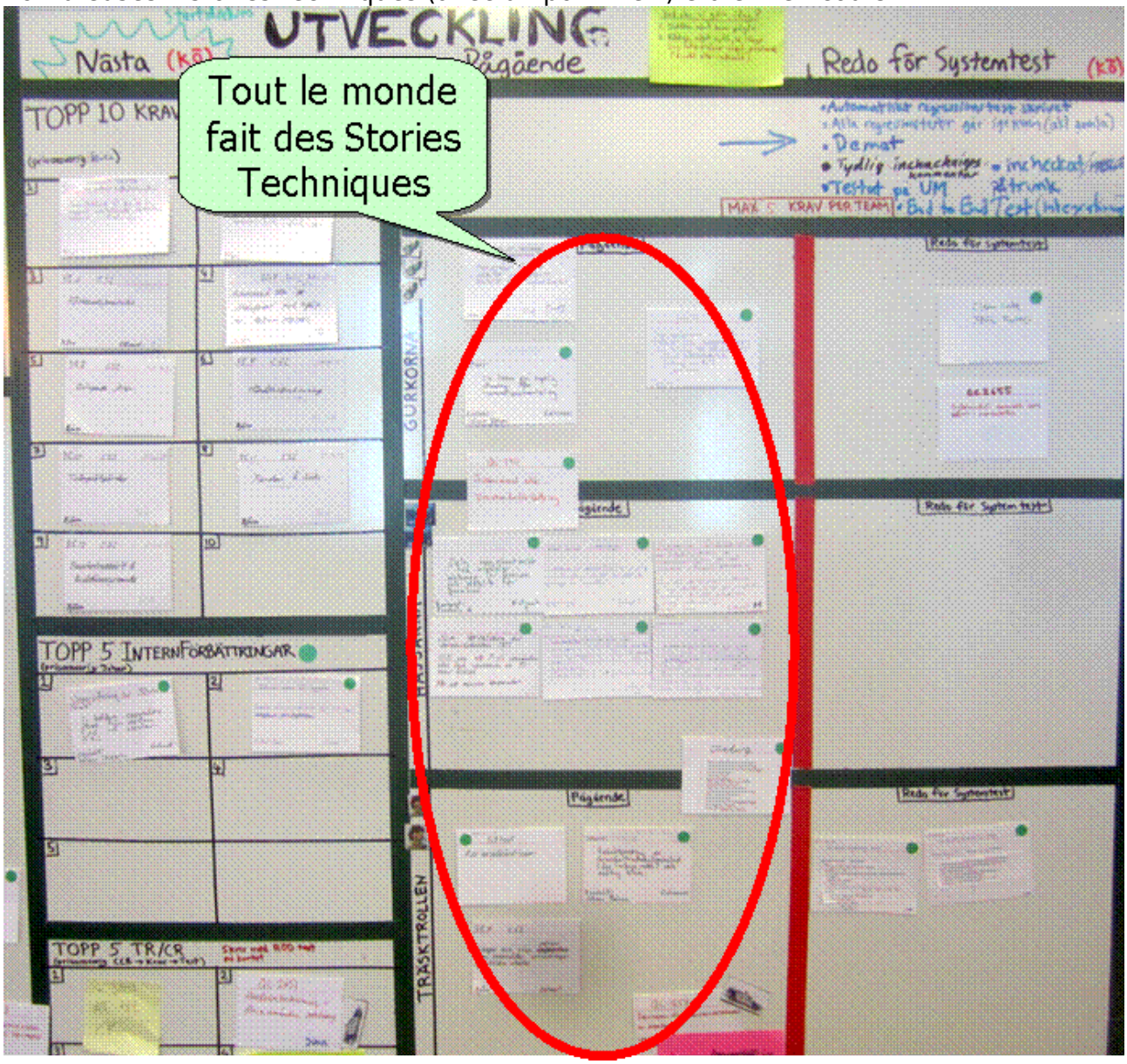
Généralement, les fonctionnalités dominent, mais voici deux exemples de situation nous ayant obligé à nous concentrer sur les histoires techniques pendant une semaine ou plus :

Exemple 1 : le test du système était devenu un goulet d'étranglement, il est donc clair qu'il n'y avait aucun intérêt à développer de nouvelles fonctionnalités pour les ajouter au goulet d'étranglement. Une fois ceci clairement établi, les développeurs se sont concentrés sur l'implémentation d'histoires techniques qui rendraient le test du système plus facile – la plupart étaient des automatisations de tests. En fait, le responsable des tests avait pour tâche de créer un backlog d'automatisation de tests, de le prioriser, d'en alimenter les développeurs via le « Top 5 des histoires techniques ». Les testeurs devinrent des clients.



Pour en savoir plus, lisez mon article « Backlog d'automatisation de test ».

Exemple 2 : C'était la veille du jour d'une livraison majeure, et l'équipe voulait sortir cette version le plus vite possible avant de commencer une nouvelle série de fonctionnalités. Elle s'est donc concentrée sur la résolution d'anomalies de dernière minute. S'il n'y avait pas d'anomalies à résoudre à ce moment, elle travaillerait sur des histoires techniques. Typiquement des points que nous voulions faire depuis longtemps mais pour lesquels nous n'avions jamais de temps, tels que supprimer du code mort, rattraper le retard sur le code à remanier, et apprendre de nouveaux outils. De nombreuses histoires techniques (avec un point vert) étaient en cours.



En aparté, je n'ai jamais vu un projet de cette taille avec aussi peu de *drames* lors des livraisons ! C'en est presque décevant :o)

Où sont passées la panique habituelle, la précipitation et la nuit blanche succédant à une journée trop courte avant la livraison ? Où sont passés le pic des demandes de support et la résolution des anomalies à chaud après livraison ? Je suis venu le lendemain de la plus importante livraison (la livraison nationale qui était le moment le

plus important de l'ensemble du projet), et il y avait à peine quelques signes témoignant que quelque chose de significatif venait de se produire.

La raison de ceci était que les livraisons étaient bien répétées, grâce aux informations recueillies lors des livraisons pilotes. Il y avait bien eu quelques problèmes avec les livraisons pilotes préalables, mais c'est pour cela que nous faisons des pilotes, n'est-ce pas ?

De toute manière, souvenez-vous de célébrer les livraisons – même si vous êtes maintenant doués pour les faire et que ce n'est plus aussi excitant qu'avant...

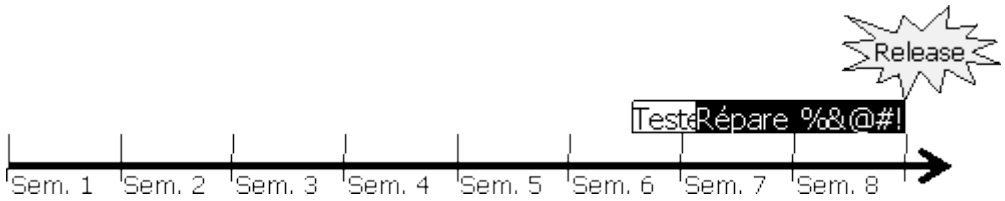
15. Comment nous avons géré les anomalies

Dans le passé, nous étions habitués à gérer les anomalies de manière traditionnelle. Les testeurs trouvaient éventuellement les anomalies pendant les tests du système à la fin du cycle de développement et les traçaient dans le gestionnaire d'anomalies. Le gestionnaire d'anomalies contenait des centaines d'anomalies, et un comité de contrôle du changement (CCC) se rencontrait chaque semaine pour analyser, prioriser, et assigner les anomalies aux développeurs. C'était un processus assez ennuyeux et inefficace.

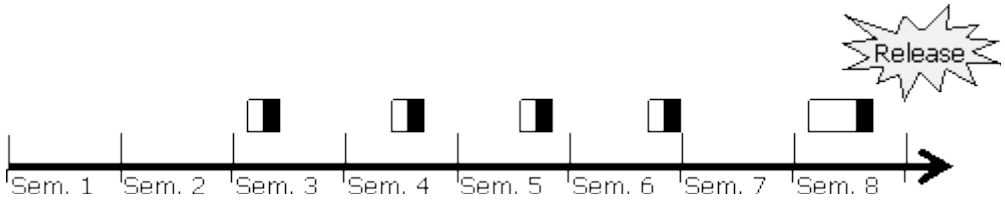
Test système en continu

La première chose que nous avons changée est la fréquence des tests du système : les faire continuellement (enfin, au moins de façon régulière) au lieu de les repousser à la fin. Au début, il y a eu quelques résistances de l'équipe de tests. En effet le test du système prend du temps et il parait inefficace de le faire plus d'une fois par cycle de livraison. Mais c'est une illusion : il semble plus efficace de tester à la fin, mais si nous rajoutons le délai de résolution dans l'équation, ça n'est plus le cas.

Voici à quoi ressemble généralement le test en fin d'un cycle de 2 mois de livraison :

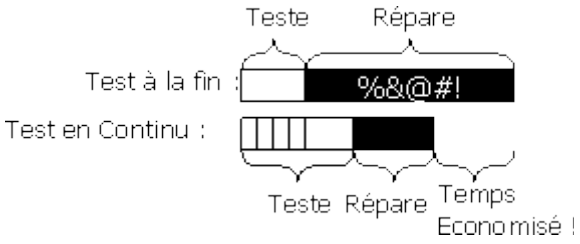


Et voici à quoi il ressemble si nous le faisons plus souvent :



Nous ne pouvons pas tester tout le système avant la fin, étant donné que le système n'est pas fini. Mais nous pouvons tout de même faire des tests du système partiel, basés sur les fonctionnalités développées durant cette période. Et nous pouvons toujours faire un test du système complet à la fin. Le test du système final peut prendre autant de temps qu'avant, mais le délai de résolution d'anomalies est réduit considérablement (et c'est la plus grosse partie). Non seulement parce que nous avons déjà trouvé beaucoup d'anomalies avant, mais aussi parce que celles que nous trouvons à la fin sont souvent des nouvelles anomalies, et par conséquent plus facile pour les développeurs à trouver et à corriger. Nous accélérons aussi l'apprentissage en détectant les anomalies plus tôt.

Mettons donc ces deux images ensemble et comparons graphiquement le temps de réalisation des tests et de détection des anomalies dans les deux scénarios :



C'est une image très importante. Regardez-là à nouveau, spécialement si vous êtes testeur. Oui, votre temps de test est plus important dans le deuxième scénario. Mais le temps total diminue !

Bien sûr, un autre élément clé est l'automatisation des tests. Nous ne pouvons jamais automatiser tous les tests, mais depuis que nous testons encore et encore le système, nous voulons automatiser autant que possible !

Corriger les anomalies immédiatement !

Désormais, lorsque les testeurs trouvent une anomalie, ils ne la tracent plus dans le gestionnaire d'anomalies. À la place, ils l'écrivent sur une étiquette rose (comme n'importe quel autre obstacle) et vont discuter avec les développeurs. Dans la plupart des cas, ils savent qui aller voir, étant donné que chaque équipe comporte un testeur travaillant avec les développeurs au quotidien. Autrement, ils demandent au leader de l'équipe et trouvent rapidement la bonne personne pour résoudre l'anomalie (généralement le développeur ayant travaillé sur cette partie du code).

Le développeur et le testeur s'assoient l'un à côté de l'autre et corrigent l'anomalie sur le champ. Il arrive aussi, quelquefois, que le développeur corrige l'anomalie de son côté et retourne voir le testeur juste après. Le point important est : jamais plus d'étapes, pas de délais ni aucune communication à travers le gestionnaire d'anomalies. Ceci est plus efficace pour plusieurs raisons :

- Trouver et corriger les anomalies plus tôt est plus efficace que de les trouver et corriger tardivement.
- Communiquer face à face est plus efficace qu'une communication écrite (en raison d'une bande passante plus étendue).
- Apprendre plus, les développeurs et les testeurs apprenant du travail des uns et des autres.
- Perdre moins de temps à gérer de longues listes de vieux défauts.

Quelquefois une anomalie n'est pas assez importante pour être corrigée immédiatement, par exemple, si c'est seulement une gène mineure pour les utilisateurs, et qu'il y a des fonctionnalités plus importantes à implémenter plutôt que corriger ce léger défaut. Dans ce cas, eh bien, oui, le testeur tracera l'anomalie dans le gestionnaire d'anomalies. À moins, bien sûr, qu'il ne soit déjà rempli.

Quoi ? Rempli ? Comment le gestionnaire d'anomalies peut-il être rempli ?

Pourquoi nous limitons le nombre d'anomalies dans le gestionnaire d'anomalies

Dans le passé, nous avions des centaines d'anomalies dans le gestionnaire. Maintenant

nous avons une limite imposée à 30.

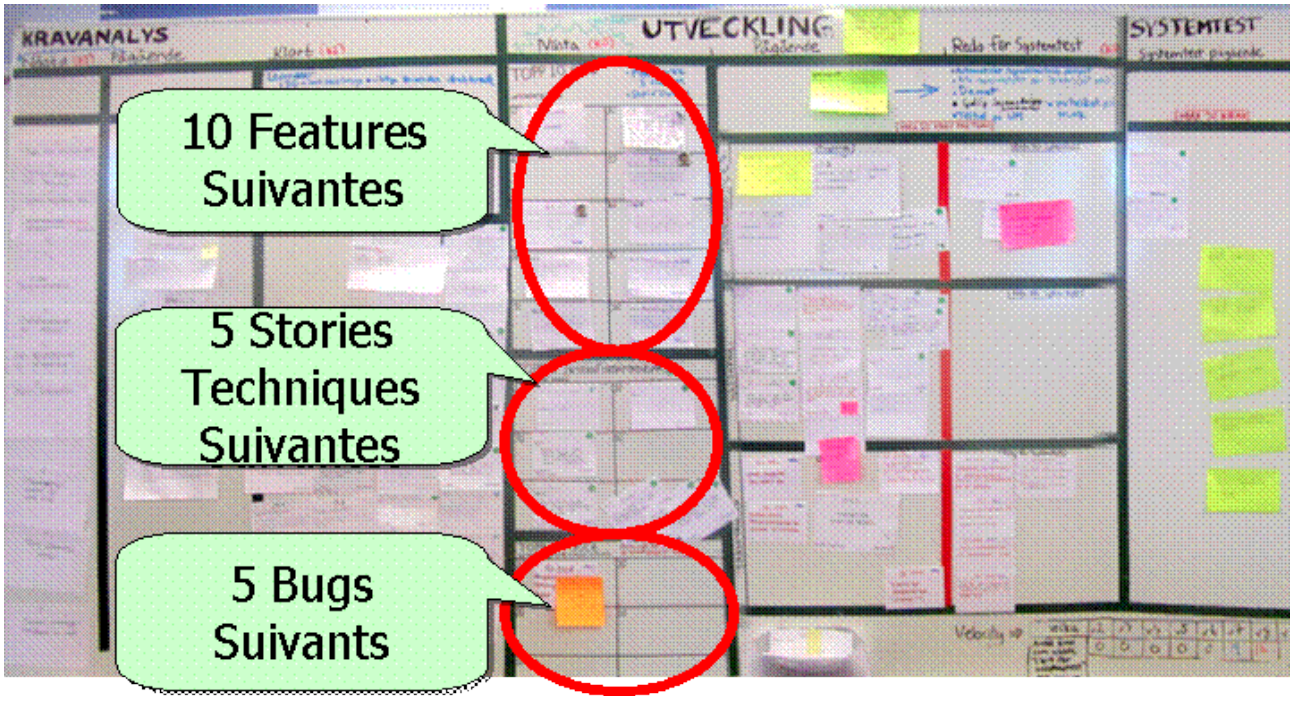
Si une anomalie est trouvée, et qu'elle n'est pas assez importante pour être corrigée maintenant, et que le gestionnaire d'anomalies contient moins de 30 éléments, alors nous ajoutons l'anomalie dans le gestionnaire d'anomalies.

Si la liste est pleine, alors il faut prendre une décision. Est-ce que cette anomalie est plus importante que n'importe laquelle de ces 30 ? Dans la négative, nous ignorons la nouvelle anomalie. Ou nous la mettons dans le gestionnaire d'anomalies avec le statut « différé » (qui serait l'équivalent de la phrase « ne sera probablement pas corrigé »). Nous faisons ceci principalement parce que cela froisse la susceptibilité du testeur de trouver une anomalie et de l'ignorer. Même si elle ne sera probablement jamais corrigée, il y a un certain réconfort psychologique de l'avoir écrite quelque part. De plus, cela peut être utile à des fins statistiques. Mais au fond, les éléments « différés » sont en dehors de la liste du top 30 dans l'équivalent d'une corbeille à papier, ou d'un sous-sol plein de choses dont nous n'avons pas besoin mais que nous n'avons pas le cœur de jeter maintenant.

Si une nouvelle anomalie est plus importante que n'importe laquelle des 30 anomalies, alors une autre anomalie est enlevée de la liste du top 30 (c'est-à-dire « différée » pour faire de la place à la nouvelle.

Comment nous visualisons les anomalies

À partir des anomalies du top 30, nous en identifions également le top 5. Celles-ci vont avec les cartes sur le tableau du projet. Cela donne une troisième file d'approvisionnement pour le développement : « 10 prochaines fonctionnalités », « 5 prochaines histoires techniques », et « 5 prochaines anomalies ».



Les cartes d'anomalies sont écrites avec un marqueur rouge, afin qu'elles soient facilement identifiables par rapport aux fonctionnalités et aux histoires techniques.

Il y a ici une importante distinction. Les anomalies importantes ne sont pas tracées dans le gestionnaire d'anomalies et ne passent pas par les « 5 prochaines anomalies » - elles sont plutôt écrites sur une étiquette rose et traitées comme n'importe quel obstacle. « Important » dans ce cas signifie « cette fonctionnalité ne sera pas livrée avec cette anomalie », ou « cette anomalie est plus importante à corriger que l'élaboration de nouvelles fonctionnalités ». Alors ne la mettons pas dans une file et corrigeons-la tout de suite.

Les éléments passant par la file des « 5 prochaines anomalies » sont des anomalies qui ne sont pas assez importantes pour être sur une étiquette rose et être corrigées immédiatement, mais suffisamment importantes pour aller dans la liste des « 5 prochaines anomalies » (généralement après avoir passé quelque temps dans la liste du top 30 du gestionnaire d'anomalies). La correction sera faite bientôt mais pas tout de suite.

Lorsque l'équipe a la capacité (généralement après avoir fini quelque chose), ses membres discutent pour déterminer s'ils prennent un des éléments du top 10 des fonctionnalités, ou du top 5 des histoires techniques ou du top 5 des anomalies.

Limiter la capacité du gestionnaire d'anomalies établit la confiance. La liste d'anomalies est courte, mais les éléments y figurant seront corrigés et n'y séjourneront pas pendant des mois sans qu'il se passe quoi que ce soit. Si une anomalie a peu de chances d'être corrigée (parce qu'elle ne rentre pas dans le top 30), c'est parce que nous avons été honnêtes sur notre manière de faire depuis le début, au lieu de générer de faux espoirs (« pas de problème, monsieur, votre anomalie est sur la liste » ... juste derrière les 739 autres...)

C'est l'idée. Ou ça s'en rapproche bien.

Un axe d'amélioration auquel nous faisons face est que nous n'avons toujours pas trouvé une manière claire et cohérente de visualiser les anomalies. Nous expérimentons encore beaucoup sur ce sujet. Les testeurs aiment avoir une image claire des anomalies en cours de correction, aussi ont-ils pour cela un tableau séparé. L'avantage est à contre-balancer avec l'inconvénient d'avoir un tableau de plus pour en garder la trace. Où vont les étiquettes ? sur le tableau d'anomalies, le tableau du projet, ou le tableau de l'équipe ? Ou devrions-nous dupliquer les étiquettes d'anomalies ? Et à propos des petites anomalies, le genre de choses prenant juste quelques minutes à corriger, comment faisons-nous pour éviter de créer trop de surcharge administrative pour tout le monde ? Beaucoup de questions, beaucoup d'expérimentations :o)

Nous avons parcouru un long chemin et avons fini par trouver une solution nous permettant de trouver et de corriger les anomalies rapidement, d'améliorer la collaboration entre les développeurs et les testeurs, d'éviter la collection de longues listes sentant le moisi dans le gestionnaire d'anomalies et ainsi que les longues et ennuyeuses réunions du CCC. Mais nous essayons encore de trouver une autre visualisation et expérimentons pour trouver le bon niveau de détail à avoir sur les tableaux.

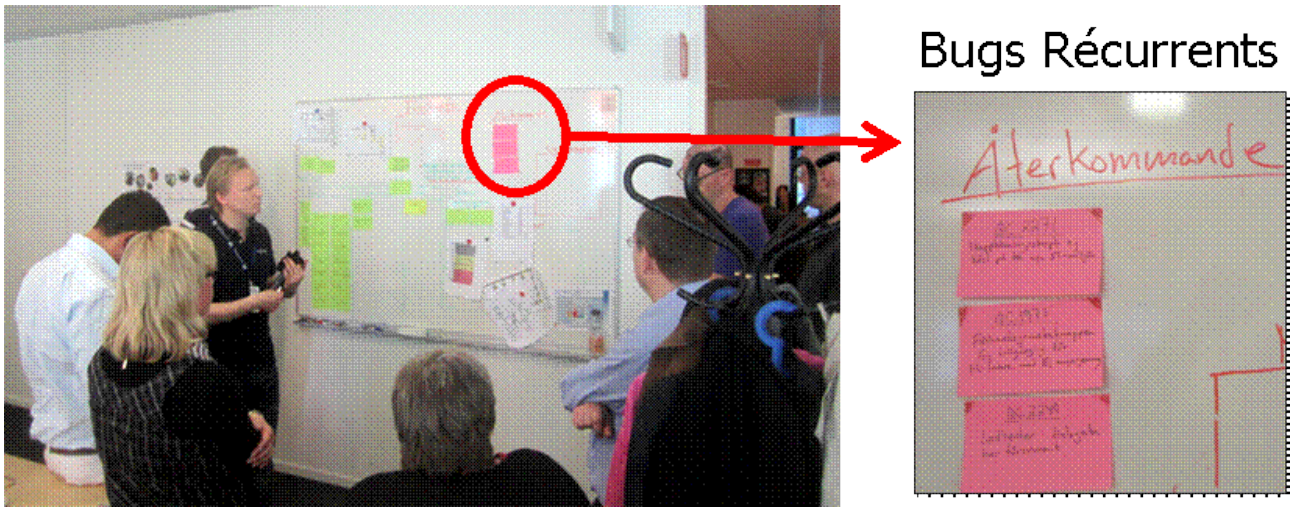
Comment nous empêchons l'apparition d'anomalies récurrentes

Certains types d'anomalies reviennent toujours. Souvent des choses simples, telles que

du texte mal aligné ou orthographié dans l'interface utilisateur. La question importante est comment ces anomalies arrivent dans le système la première fois, quel problème sous-jacent du processus cause le problème technique ?

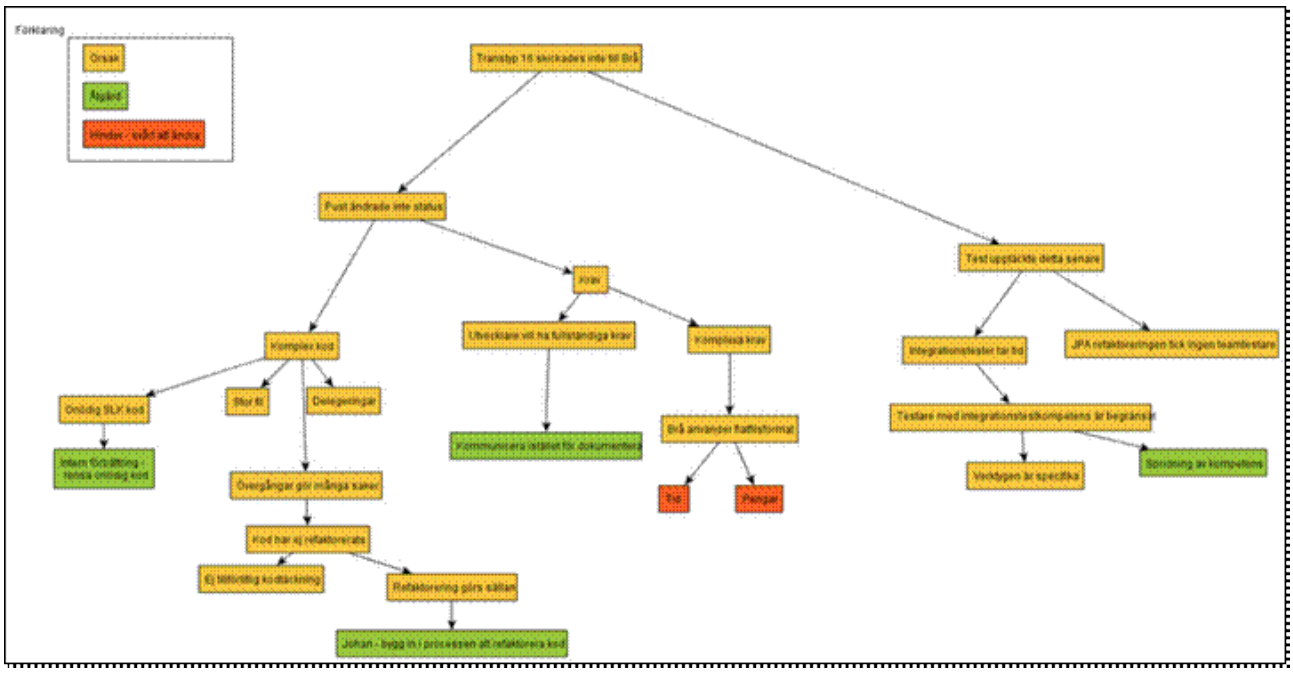
Pour aider à corriger ce problème (au lieu de se plaindre seulement) les testeurs ont une section de leur tableau appelée « anomalies récurrentes ». Est-ce que vous vous rappelez que je disais que les anomalies étaient écrites sur des étiquettes roses et traitées comme n'importe quel obstacle ? Eh bien, lorsque les testeurs sentent qu'une anomalie spécifique leur donne un fort sentiment de déjà-vu, ils la mettent sous la section « anomalies récurrentes » de leur tableau. Elles se limitent à une poignée.

(Vous avez dû remarquer l'idée principale maintenant, hein , Limitez les files !)



Quelquefois, les équipes de développement ont une réunion de prévention des anomalies, où ils prennent les anomalies récurrentes et font une analyse des causes racines. Comment cette anomalie est-elle arrivée dans le code ? Quels sont les problèmes systémiques qui nous ont fait créer de manière répétitive ce type d'anomalie, et que pouvons nous faire à ce propos ? Parfois cela est en rapport avec les outils que nous utilisons, d'autres avec les règles et les marches à suivre, des problèmes culturels, etc.

Un diagramme cause-effets est généralement utilisé pour identifier les conséquences et les causes racines de cette anomalie, qui est alors utilisé pour générer des réponses concrètes pour déterminer comment éviter ce type d'anomalies dans le futur.



Les diagrammes causes-effets sont un excellent moyen de faire de l'analyse de cause racine, notamment lorsque vous commencez à tomber sur des cercles vicieux. Par exemple « Cette anomalie a été causée par le stress et en ne testant pas correctement. Nous étions stressés parce que nous étions en retard par rapport au calendrier de livraison. Nous étions en retard sur le calendrier de livraison parce que nous avions des anomalies dans la livraison précédente ». Une boucle assez vicieuse n'est-ce pas ?

Jerry Weinberg le résume de manière élégante : « Les choses sont ainsi parce qu'elles ont été faites ainsi ». Ou dit autrement (j'ignore qui l'a dit en premier) : « si tout ce vous faites toujours est tout ce que vous avez toujours fait, alors tout ce que vous aurez toujours sera tout ce que vous avez toujours eu ».

De toute manière, si vous voulez en apprendre plus sur les diagrammes causes-effets, lisez mon article « diagrammes causes-effets – une manière pragmatique de faire de l'analyse des causes racines ».

16. Comment nous avons continuellement amélioré le processus

Le processus décrit dans ce papier n'a, en aucune manière, été conçue à l'avance. Je n'aurais jamais été capable de tout concevoir au préalable, surtout seul. Et même si j'avais pu, il n'y aurait pas eu d'adhésion de la part des participants au projet, en fait tout processus que j'aurais conçu préalablement n'aurait jamais dépassé le stade du papier sur lequel il aurait été écrit.

Le processus a d'ailleurs été découvert plutôt que conçu. La première chose que j'ai faite a été de mettre en place un « moteur d'améliorations du processus ». En réalité, je n'utilisais pas ces mots mais c'est pourtant de cela dont il s'agit. La formule basique est :

- **Clarté** – avoir des tableaux physiques dans des lieux stratégiques pour montrer à tous ce qu'il se passe. Avoir un objectif de livraison clair, que tout le monde puisse comprendre.
- **Communication** – avoir des ateliers hebdomadaires d'amélioration du processus au niveau de chaque équipe et à un niveau transverse.
- **Données** – utiliser quelques métriques simples nous montrant si notre processus s'améliore. Nous mesurons la vélocité et le temps de cycle. Voir le chapitre 19 pour plus d'informations.

Cette stratégie est assez simple, elle est basée sur le postulat que les gens, par nature, désirent réussir dans leurs projets, aiment résoudre les problèmes et améliorer leurs façons de travailler. Nous avons créer un environnement permettant et encourageant ces comportements.

Si tout le monde sait où nous allons, où nous sommes actuellement et qu'il existe de bons espaces de communications en place, alors il y a des chances que les gens s'organisent pour aller dans la bonne direction en trouvant continuellement des manières d'y aller plus vite.

Motiver les gens à améliorer leur processus évolutif est à la base de l'Agilité et du Lean.

Comment nous avons fait les rétrospectives d'équipes

Nos ateliers d'améliorations de processus sont fondamentalement des rétrospectives de fin de sprint Scrum. Par convention, les équipes utilisent le mot « rétrospective » pour leurs ateliers d'améliorations de processus, et l'équipe transverse utilise le terme « atelier d'améliorations de processus », aussi me suis-je attaché à utiliser cette terminologie dans ce papier.

Les équipes ont des rétrospectives toutes les semaines ou toutes les deux semaines, la durée variant entre 30 minutes et 2 heures. Quelques équipes se tiennent juste debout devant leur tableau et font la rétrospective, d'autres vont dans une salle séparée. Généralement, le leader de l'équipe facilite la réunion, mais quelquefois quelqu'un d'autre est appelé (comme moi) à remplir ce rôle. Avoir de temps en temps un facilitateur externe à l'équipe est généralement une bonne idée, cela donne à l'équipe des variations sur la manière dont la rétrospective se déroule, et fournit également au

leader de l'équipe d'autres idées sur la façon de mener les rétrospectives. Cela lui permet aussi de participer pleinement au lieu de faciliter.

Une manière simple et bon marché d'obtenir un facilitateur externe est d'échanger entre équipes : le leader de l'équipe A facilite la rétrospective de l'équipe B, et réciproquement.

Il y a une assez grande variation sur la manière dont les rétrospectives sont menées, mais l'objectif est le même dans chaque cas : réfléchir sur ce qui fonctionne bien et ce qui ne fonctionne pas, et décider quoi changer.

Les changements typiques portent sur des choses comme déposer du code plus souvent, changer l'heure de la réunion quotidienne ou comment mener la réunion, mettre à jour les conventions de codage, identifier des nouveaux rôles internes à l'équipe, etc...

Une autre fonction importante des rétrospectives de niveau équipe est d'identifier les points d'escalade, c'est-à-dire les problèmes et les propositions d'améliorations affectant non seulement cette équipe mais également d'autres et dont les résolutions ont besoin d'être faites avec ces autres équipes. Tout ceci est noté par les leaders d'équipe et rapporté aux ateliers d'améliorations de processus de plus haut niveau.

Comment nous faisons les ateliers d'améliorations de processus

L'atelier d'amélioration du processus est fondamentalement une rétrospective de type « Scrum de Scrums », avec une personne de chaque équipe technique et de chaque discipline (la même équipe transverse se réunissant devant le tableau projet tous les jours à 10:00). En tant que coach, une de mes tâches les plus importantes était de mettre en place ce forum et de le faciliter. C'est le lieu le plus important pour déclencher le changement et découvrir quels changements fonctionnent.

L'objectif déclaré de cette réunion est de clarifier et d'améliorer notre façon de travailler.

Initialement, nous la faisions tous les jeudis à 13h. La faire les jeudis à 13h était une coïncidence – c'était le moment le moins encombré pour les personnes impliquées. Après environ un mois, nous sommes passés à une fréquence d'une semaine sur deux, soit un jeudi sur deux à 13h. La raison pour laquelle nous le faisions aussi souvent au début était dû au besoin d'améliorer rapidement la collaboration entre les différentes disciplines, et au besoin urgent de changer, provoqué par l'augmentation des difficultés et par la confusion dans laquelle nous étions.

Avoir des ateliers d'amélioration du processus chaque semaine, c'ést plutôt intense, il y avait peu de temps pour appliquer les changements d'une réunion à l'autre. Le côté positif était que cela nous obligeait à implémenter le changement rapidement, parce que c'était embarrassant d'être assis à l'atelier d'amélioration du processus suivant et dire « eh bien, pris au piège, nous ne sommes finalement jamais arrivé à implémenter ce changement ». De plus, les réunions étant chaque semaine, nous devions les maintenir courtes et focalisées, ce qui nous forçait à prioriser uniquement les changements importants et gravir de petites marches dans le processus du changement.

En fait, en y repensant, les réunions n'étaient pas si courtes que cela. Nous avons commencé par 60 minutes et nous avons dû passer à 90 minutes parce que nous n'arrêtions pas de déborder. Une durée plutôt longue pour une réunion hebdomadaire. Et les changements que nous faisions étaient assez conséquents, pas du tout des petits pas de bébés. En regardant en arrière, je ne peux pas vraiment dire si cela était une bonne chose ou non. Nous avions vraiment besoin de changer les choses assez rapidement (notamment et pas seulement, à cause de l'Horrible Grosse Date Butoir rôdant systématiquement autour de nous), mais le rythme du changement a aussi provoqué quelques confusions, spécialement dans la majorité des personnes qui n'étaient pas dans l'équipe transverse, et qui voyaient beaucoup de changements se produire sans avoir eu la possibilité de les comprendre ou d'en discuter.

Une fois que les problèmes les plus importants ont été réglés, le rythme du changement a pu être ralenti à un niveau plus confortable, aussi avons-nous réduit le nombre de réunions à une toutes les deux semaines. Cela a été ressenti comme plus humain. Maintenant nous pouvons passer 90 minutes sans ressentir de stress (ce n'est plus chaque semaine) ; il est plus facile d'implémenter réellement le changement et d'apprendre de ses effets avant la prochaine réunion.

Lorsque j'animais les ateliers d'amélioration de processus, je faisais attention d'éloigner toutes les tables pour faire un cercle des chaises au centre de la pièce à côté d'un tableau blanc.

Cela a eu un effet notable sur le niveau de collaboration et d'attention dans la pièce. Chacun faisant face à l'autre sans aucune barrière entre eux, et sans sources de distraction comme des papiers ou des ordinateurs sur la table.

Chaque atelier d'améliorations du processus suit la même structure basique, correspondant grossièrement à la structure des réunions définies dans le livre « Agile Retrospectives » de Diana Larsen et Esther Derby.

Globalement :

- **Préparer le terrain** : Ouvrir la réunion en indiquant le thème et le point spécifique
- **Rassembler les données** : Examiner ce qui s'est passé depuis la dernière réunion, que ce soient les victoires ou les points difficiles. S'il y a un thème,

examiner les données réelles en rapport avec ce thème
- **Générer des idées** : Discuter des données et de ce qu'elles signifient pour nous, se focaliser sur les points difficiles les plus importants et identifier les options concrètes pour les résoudre
- **Décider quoi faire** : Prendre des décisions sur le choix des changements à implémenter
- **Clore la réunion** : Décider qui va faire quoi, et ce qu'il va se passer d'ici la prochaine réunion.

Je commence par faire un rapide tour de table de sorte que tout le monde puisse parler. Par exemple « Quel est votre sentiment maintenant, en utilisant 1 ou 2 mots » ou « quelle est la chose la plus importante que vous espérez obtenir en sortant de cette réunion ».

Je rappelle, alors à tout le monde l'objectif de la réunion et mentionne quel va être le point spécifique de la réunion du jour. Quelques fois nous avons un point spécifique (tels que la motivation, les métriques, la collaboration, l'automatisation des tests, ou d'autres sujets). Le reste du temps, l'objectif est celui, général, d'améliorer notre façon de travailler.

Alors, nous résumons les évènements clés de la période passée (c'est-à-dire le temps écoulé depuis la dernière réunion) et revoyons les décisions et actions de la dernière réunion pour examiner comment cela s'est passé.

Puis, nous regardons brièvement ce qui marche bien et les développements globalement positifs durant ces dernières semaines. Parfois, les participants écrivent sur des notes adhésives et les collent au mur. D'autres fois ils ne font que parler et j'écris au tableau blanc. Noter et célébrer l'amélioration est important pour motiver l'arrivée d'autres améliorations.
Ensuite, nous résumons brièvement les points difficiles et les défis. S'il y en a plusieurs (ce qui est généralement le cas), nous faisons une sorte de priorisation, généralement à points fixés ou quelque chose comme cela. Voter à points fixés signifie que chaque personne dans la pièce a 3 points qu'il distribue à sa convenance, selon l'importance perçue, sur l'ensemble des éléments.

Voici un exemple de deux colonnes de notes adhésives : « victoires » et « défis ».

<digression>
N'importe quel enthousiaste de lean du type le plus extrême lisant cet article pourrait pointer un doigt accusateur maintenant et dire « Beurk, c'est complètement subjectif, un machin gentillet ! L'amélioration d'un processus doit être conduite par des rapports, des données quantitatives et objectives ! »

Eh bien, tout d'abord, je ne suis pas d'accord. Le développement de produit complexe de cette sorte est un processus créatif fait par des personnes créatives, et le paramètre le plus important est la motivation. Dans ce contexte, les tripes parlent plus que des métriques réelles. Si quelque chose est ressenti comme un problème important, c'est certainement un problème important, que nous ayons ou pas les métriques pour le prouver. Et un truc sympa avec les tripes, c'est que cela est souvent un indicateur d'un problème qui se rencontrera prochainement, alors que les métriques réelles montrent souvent un problème seulement après qu'il se soit produit.

Alors oui, nous utilisons des métriques réelles (décrites plus bas dans ce papier) et quelquefois ces métriques indiqueront la réalisation de la prémonition par les tripes, qu'il y a un problème (ce moment où on se dit « oh merde »). Mais nous utilisons les métriques réelles avant tout pour soutenir notre amélioration de processus, pas pour la mener.
</digression>

Bon, après tout cela, où en étions-nous ? Ah oui, nous listions les points difficiles et les

priorisions et choisissions 1 ou 2 sujets sur lesquels nous nous focalisions pendant la réunion. Ensuite, généralement, nous nous séparions en groupes de 2 ou 3 pour discuter et analyser les problèmes et les solutions possibles.

Pour des problèmes plus complexes ou récurrents, nous faisions une analyse des causes racines en utilisant les diagrammes causes-effets et d'autres techniques similaires, ou nous proposions quelques recherches amenant des métriques utiles au prochain atelier d'améliorations de processus, ou nous planifiions un atelier séparé de résolution de problème à faire avec une petite équipe se concentrant dessus. Néanmoins, la plupart des problèmes ne sont pas traités de cette manière là.

La discussion en sous-groupes fournit généralement un nombre de propositions concrètes, ou d'options que je liste sur tableau blanc. L'option par défaut est toujours le statu quo (« ne rien changer »), un rappel de ce qu'il se passera si nous ne sommes pas d'accord sur une autre option d'ici la fin de la réunion.

Pour chaque option (y compris l'option du statu quo), nous réfléchissions sur les avantages et inconvénients évidents. Assez souvent cette analyse rapide révèle clairement quelle option est la meilleure, aussi sommes-nous rapidement d'accord pour l'implémentation de cette option. Pour des choix moins évidents, nous faisions un rapide vote avec le pouce, c'est-à-dire vérifier comment les gens ressentent chaque option.

- Pouce levé signifie « J'aime cette option ».
- Pouce sur le côté signifie « Cette option n'est pas géniale, mais acceptable ». Ou je n'ai pas une opinion tranchée et je rejoins le groupe ». Ou « Je ne peux me faire une opinion tout de suite ».
- Pouce en bas signifie « Cette option est nulle à chier, et je ne la soutiendrai pas ».

Quelquefois, nous utilisons « le poing de 5 » à la place, qui est globalement la même chose mais avec une granularité plus fine. À la place d'utiliser uniquement le pouce levé, sur le côté ou en bas, chaque personne lève entre 1 et 5 doigts.

- 5 = C'est une option géniale !
- 4 = C'est une assez bonne option.
- 3 = (même chose que le pouce sur le côté).
- 2 = Je n'aime pas cette option et je ne la soutiendrai pas. Mais je peux être convaincu.
- 1 = Passez-moi plutôt sur le corps.

La chose importante avec ces techniques est que le pouce sur le côté, ou les 3 doigts, est une valeur-palier signifiant « Je soutiendrai cette option ». Aussi, toute option ayant un niveau de soutien équivalent (ou supérieur) de la part de chaque personne de l'atelier est bonne à être adoptée. Tout le monde n'a pas à aimer cette option, mais tout le monde la soutiendra. C'est ce que le consensus signifie.

En examinant les options et les votes, il devient généralement clair quelle est la meilleure option. S'il y a trop d'options, nous commençons par éliminer les options inacceptables, c'est-à-dire les options n'ayant que des votes à 1 ou 2 doigts ou le pouce en bas, car ces options n'auront pas le consensus du groupe. Un peu quelque chose comme un veto. Ensuite, nous prenons les options restantes et sélectionnons celles ayant les votes les plus élevés. De nouveau, ce sera le statu quo par défaut si nous ne pouvons pas être d'accord sur une option.

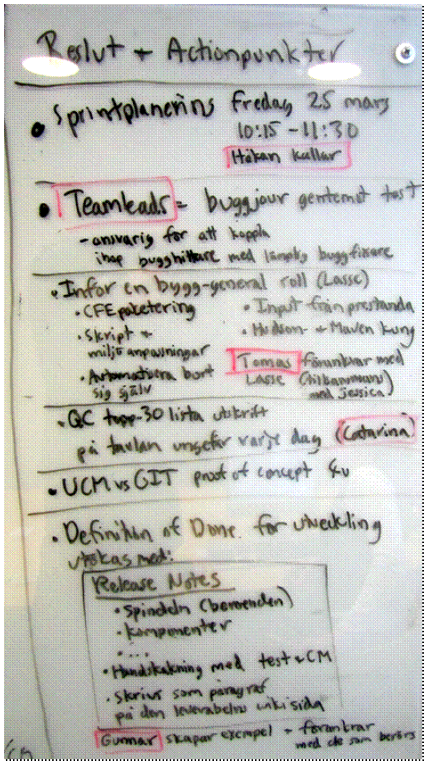
Dans les rares cas où il y a deux options ayant le plus haut niveau de soutien identique, nous en sélectionnons une au hasard, la plus facile, ou faisons un vote à la majorité (« étant donné le choix entre les options D et E, quelle est celle que vous préférez »). En tant que facilitateur de la réunion, je décide généralement du processus de décision dans chaque cas, pour éviter de perdre trop de temps en méta-discussions à propos de comment décider du processus de décision (c'est toujours un risque avec les gens intelligents). C'est aux participants de la réunion de protester s'ils n'apprécient pas mes choix.

Tout ce travail d'élaboration du consensus peut sembler contre productif, mais il est en réalité assez rapide et efficace dans la plupart des cas. Et dans le peu de cas où cela n'est pas rapide, cela signifie qu'il y a besoin d'un peu plus d'analyse en profondeur.

Les décisions que nous prenons lors de ces réunions d'amélioration de processus sont généralement, eh bien, des décisions d'amélioration de processus. Cela signifie du changement. Et comme nous avons à faire à des Personnes, changement signifie risque de résistance. Spécialement avec ceux qui n'étaient pas présents à la réunion où les décisions ont été prises. En visant 100% de consensus pour chaque changement (à l'intérieur de l'équipe transverse présente à la réunion) nous réduisons de manière spectaculaire le risque de la résistance, et augmentons aussi la probabilité que le changement se passera bien. Ainsi, les quelques minutes supplémentaires passées à créer le consensus sont rentables au-delà de nos espérances.

Nous limitons strictement la durée de la réunion, généralement à 90 minutes. Durant 10 minutes environ, nous résumons les décisions qui ont été prises (listées sur le tableau blanc) et identifions les actions concrètes – qui va faire quoi et quand.

Voici un exemple :



Au cours de cette réunion, nous avions pris beaucoup de décisions (plus que d'habitude). Les deux premières sont :

- Essayer le concept des « réunions de planification de sprint », où les différentes disciplines collaborent pour raffiner ou éclater les histoires d'utilisateurs et décident lesquelles tirer dans la colonne des « 10 prochaines fonctionnalités » du tableau. Il y a d'autres d'informations sur ce sujet plus loin dans ce papier. Si cela fonctionne bien, nous continuerons probablement une semaine sur deux.
- Chaque équipe a un « référent pour les anomalies » clairement identifié, c'est-à-dire la personne par défaut à qui les testeurs vont parler lorsqu'ils ont trouvé une anomalie. Le leader de l'équipe est le référent par défaut pour les anomalies si personne d'autre n'est désigné.

Rappelez-vous que l'objectif affiché de la réunion est de clarifier et d'améliorer le processus actuel. Quelquefois, en fait, nous ne changeons rien, et à la place, nous clarifions juste notre processus actuel, c'est-à-dire résoudre certaines sources de confusions et apportons une description claire que chaque personne à la réunion pourra relayer à son équipe. Un exemple a été le besoin de clarifier ce que nous voulions dire par « test d'acceptation » contre « test du système » contre « test fonctionnel ».

Comment nous avons géré le rythme du changement

Les réunions hebdomadaires d'amélioration du processus ont provoqué une rafale de changements, la plupart très utiles. Toutefois, après un moment, nous avons réalisé que nous changions trop et trop vite.

C'était un problème intéressant, dans la plupart des organisations où j'ai travaillé, le problème était qu'il y avait trop peu de changements de processus, tout le monde restait immobilisé dans ses processus inefficaces. Là, nous avions le problème opposé. Nous avions fait beaucoup de changements, et cela prend du temps pour que le changement de processus se propage sur un projet de 60 personnes. Beaucoup de membres de l'équipe ont été troublés (et quelques fois frustrés) lorsque l'équipe transverse a introduit de nouveaux changements avant que la poussière des changements précédents ne retombe.

Aussi avons-nous introduit un petit peu de bureaucratie pour ralentir le rythme du changement :o)

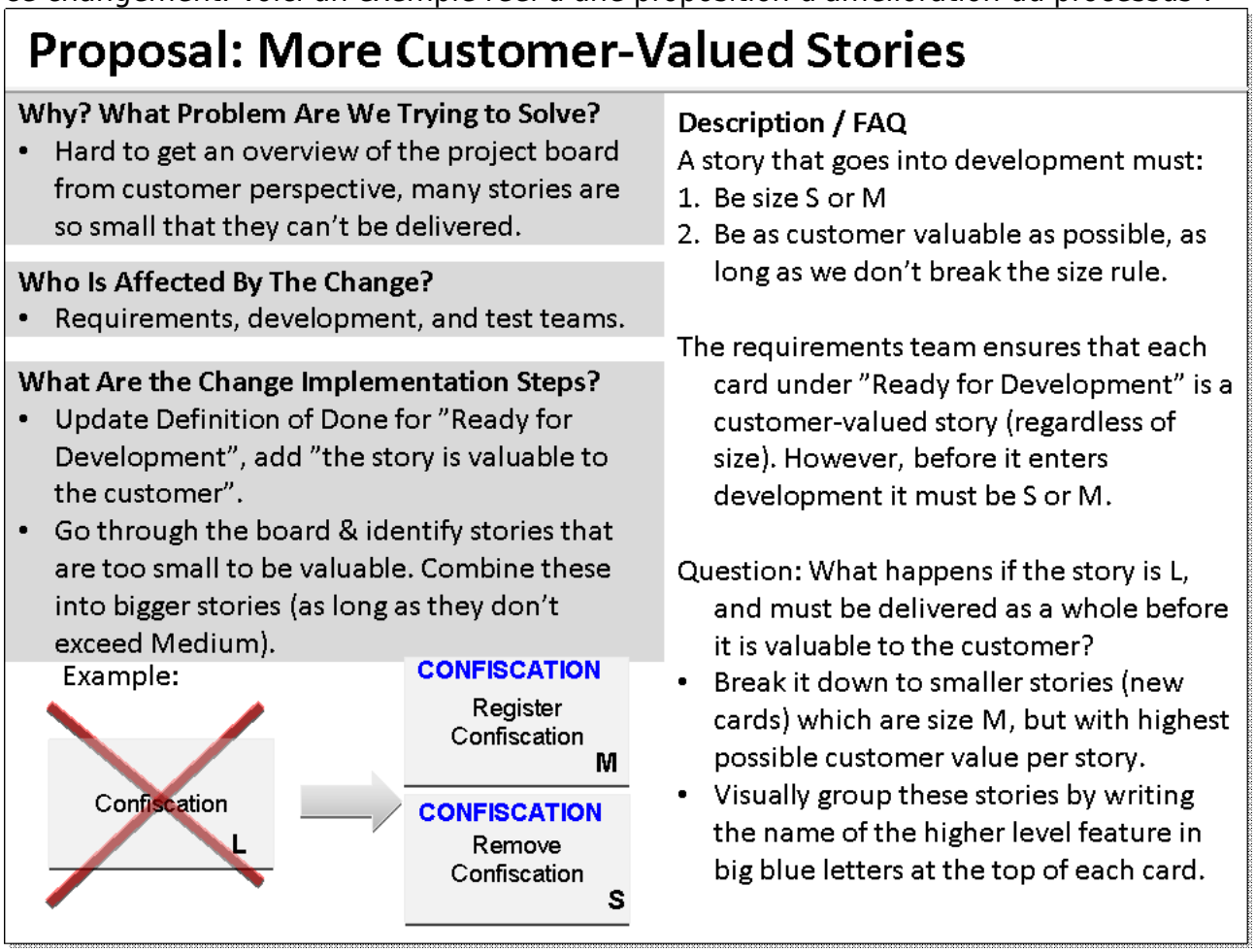
Lorsque quelqu'un veut faire un changement qui aura un impact au-delà de sa propre équipe, il écrit une « Proposition d'amélioration du processus ». C'est une variante plus légère qu'un processus de résolution de problème A3 (voir notre modèle A3 pour plus de détails sur le sujet).

Le modèle de proposition d'amélioration de processus vous force à réfléchir à la Raison pour laquelle vous souhaitez faire ce changement. Les trois questions sur le modèle sont :

- « Quel problème essayez-vous de résoudre ? »
- « Qui est affecté par ce changement ? »
- « Quelles sont les étapes nécessaires dans l'exécution de ce changement ? »

Les réponses à ces questions sont très utiles pour déterminer le rapport coût / valeur de

ce changement. Voici un exemple réel d'une proposition d'amélioration du processus :



La proposition était de conserver les fonctionnalités à un niveau apportant de la valeur pour le client. Elle proposait, aussi, que les fonctionnalités estimées trop grosse ne devraient pas être tirées en développement du tout, car elles tendaient à envahir et à bloquer le processus. À la place, elles devaient être découpées en livraisons plus petites. Et ceci fait, si les petites fonctionnalités indépendamment n'avaient pas de valeur pour le client, alors un titre devait être écrit en haut de la carte en texte bleu gras, montrant qu'un nombre de petites fonctionnalités s'assemblaient en une grosse fonctionnalité. Ceci nous aidant à garder les fonctionnalités ensemble d'un point de vue de livraisons.

Ce type de propositions peut venir de n'importe qui. Généralement, la personne ayant écrit la proposition vient à la réunion d'amélioration du processus pour présenter sa proposition et répondre aux questions. Notre modèle transforme la proposition en une mini analyse de rentabilité pour un changement spécifique, la rendant assez évidente pour prendre la décision.

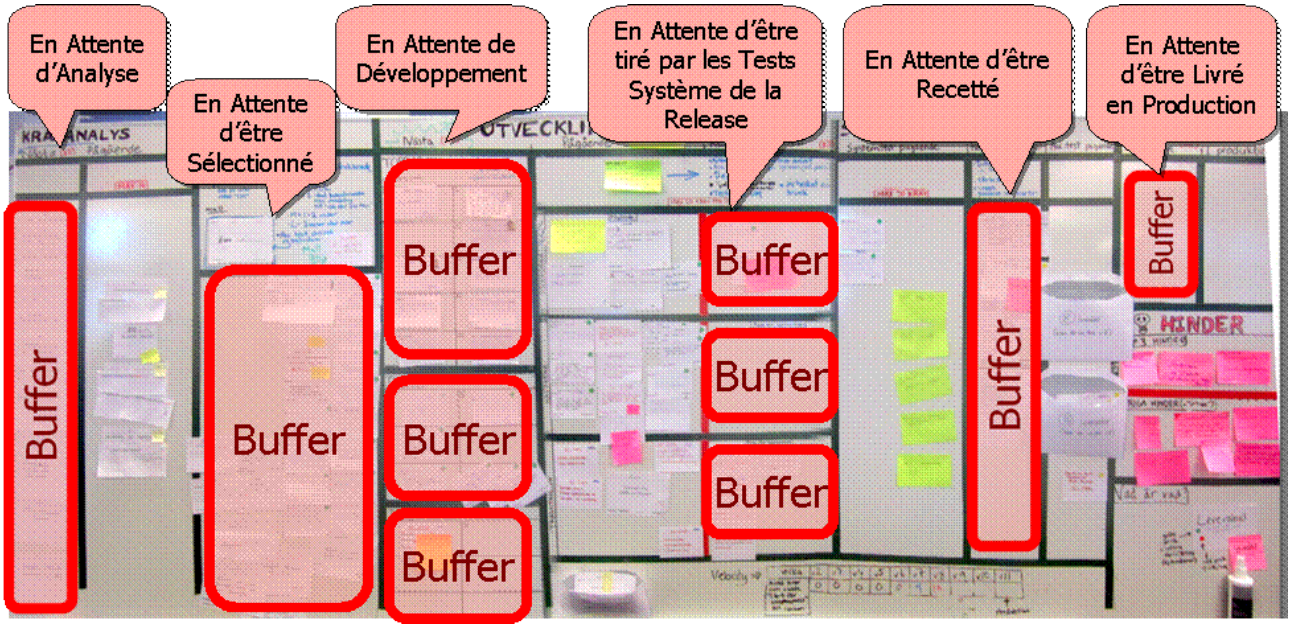
L'objectif d'introduire ce petit modèle est de nous permettre de limiter le nombre de changements. Ainsi, si nous avons 4 propositions, nous pourrions n'en implémenter qu'1 ou 2 d'entre elles, même si les 4 sont géniales. C'est très difficile de ne * pas * implémenter une proposition d'amélioration de processus géniale, mais nous avons réalisé que nous devions vraiment limiter le nombre d'initiatives simultanées d'améliorations du processus, sinon nous risquions de créer une confusion pénalisant les

bénéfices de l'amélioration.

Nous avons même considéré avoir un tableau Kanban d'amélioration du processus avec une limite du TAF, montrant quels changements étaient en cours d'implémentation. Cela pouvait être utile dans des objectifs de suivi. Mais cela aurait été un tableau de plus, pour lequel il aurait fallu trouver de la place, et à mettre à jour. Hmmmm.

17. Comment nous faisons la distinction entre les buffers et le travail en cours.

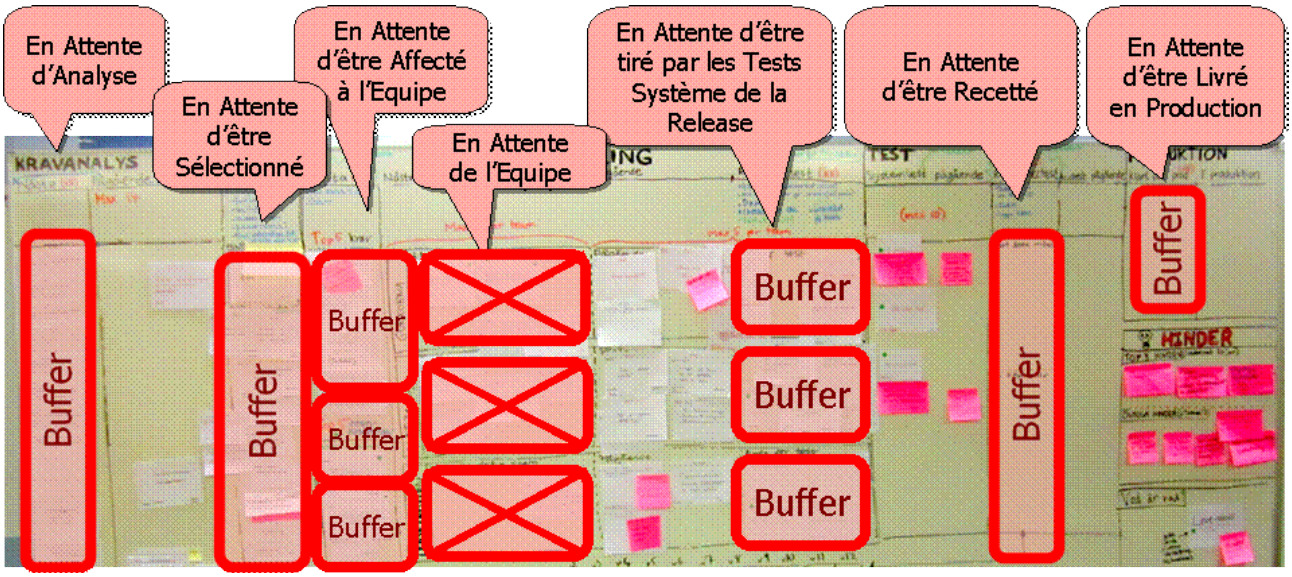
Quatre des colonnes du tableau de projet représentent du travail en cours, alors que les 6 autres sont plutôt des buffers (ou des files d'attente), dans le sens où le travail dans ces colonnes n'est pas réellement en cours – il attend de passer à la prochaine étape du processus.
Nous les mettons en évidence en les marquant en rouge, voilà ce que ça donne :



Il s'avère très utile de faire la distinction entre ces colonnes et le travail en cours, parce que les files et buffers sont clairement du gaspillage dans le sens où les fonctionnalités placées là sont seulement en train d'attendre. Une partie de ces buffers est généralement nécessaire pour absorber la variabilité, par exemple pour prendre en compte le fait que le rythme auquel les fonctionnalités sont développées ne correspond pas toujours à celui auquel elles sont testées au niveau du système. Cela peut être considéré comme un « gaspillage nécessaire » pour assurer un flux régulier.

Au fur et à mesure que le processus s'améliore, le besoin d'avoir ces buffers diminue. Il est cependant utile de les visualiser sur le tableau, pour continuer à se demander si on en a réellement besoin et ce qui peut être fait pour les réduire.

Voilà un exemple d'une ancienne version du tableau, à une époque où nous avions un autre buffer entre les exigences et le développement :



La raison d'être de ce buffer était que, à cette époque, les équipes suivaient un modèle plus orienté Scrum, avec des sprints. Ainsi à chaque réunion de planification de sprint d'une équipe, celle-ci s'engageait sur un ensemble défini de fonctionnalités. Nous avions alors 3 files (ou buffers) entre analyse et développement :

- les fonctionnalités identifiées pendant l'analyse, mais pas encore sélectionnées dans la liste du top 10
- les fonctionnalités incluses dans le top 10, mais pas encore prises par une équipe
- les fonctionnalités dans le sprint courant de l'équipe X, mais pour lesquelles le travail n'a pas encore commencé.

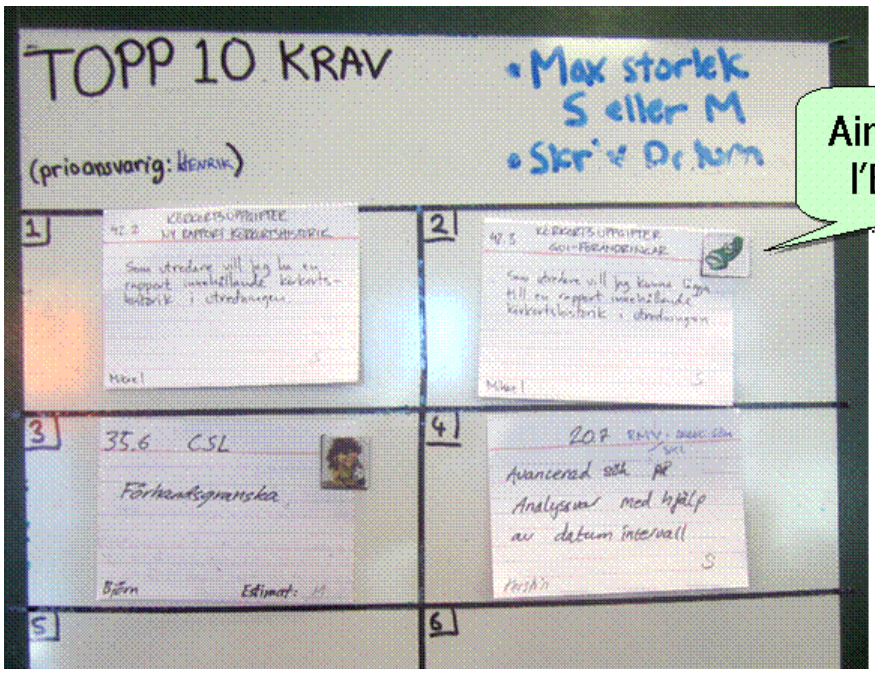
Nous avons noté que nous passions du temps à discuter de quelle fonctionnalité devrait être dans quelle file, ce qui était un gaspillage notable. Nous avons donc simplement éliminé la 3ème file et décidé que chaque équipe prendrait les fonctionnalités directement dans la liste top10 au lieu de répartir les fonctionnalités dans les sprints.

Cela soulève la question de la spécialisation. Si une équipe travaille dans un domaine spécifique à une fonctionnalité, par exemple intégration avec le système X, alors il paraît plus efficace que cette équipe réalise les fonctionnalité qui s'intègrent avec le système X, parce qu'elle a la connaissance de comment le système X fonctionne.

Cela ne signifie pas, cependant, que l'équipe a besoin de prendre *toutes* les fonctionnalités liées à X dès le début. Bien que cette équipe puisse être le choix par défaut pour toutes les fonctionnalités en relation avec X, nous ne voulons pas écarter la possibilité que d'autres équipes y contribuent, pour éviter que cette équipe devienne un goulet d'étranglement.

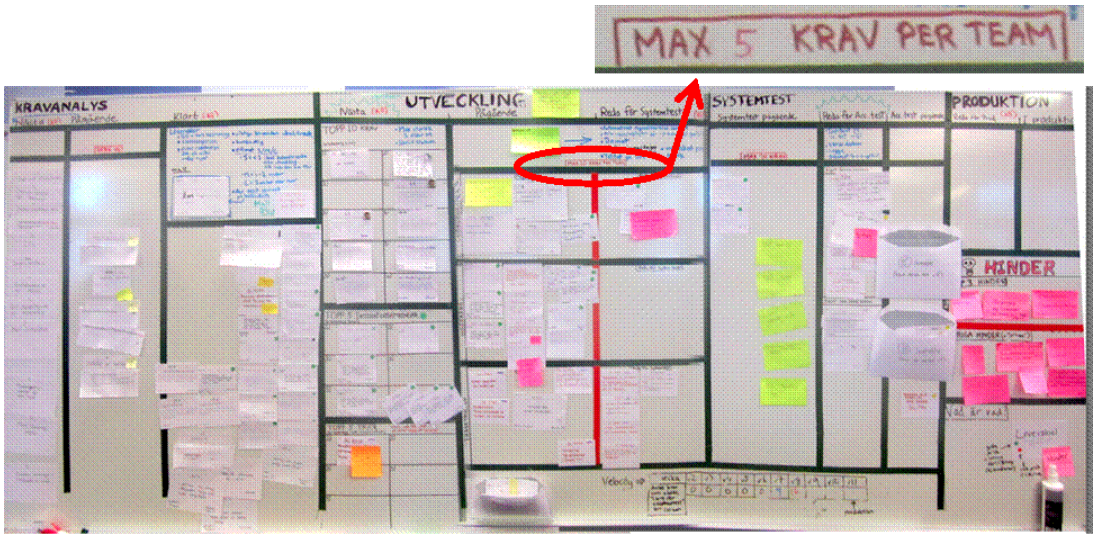
Pour résumé simplement, chaque équipe prend directement dans la liste top10, mais de façon intelligente : les équipes se parlent pendant la réunion de synchronisation et trouvent comment utiliser au mieux la capacité de chacune lorsqu'elle devient disponible.

Pour nous aider à suivre ce processus, nous avons des «aimants d'équipe » (des magnets) sur le tableau du projet, c'est-à-dire qu'une équipe « taggue » une fonctionnalité dans le top 10 (ou plus tôt) avec son aimant d'équipe , pour indiquer que « ce serait mieux que cette fonctionnalité soit réalisée par notre équipe ». De cette façon les gens savent à qui parler au sujet de cette fonctionnalité et les autres équipes y réfléchiront à 2 fois avant de la commencer.


TOPP 10 KRAV
(prioansvarig: HENRIK)
• Max storlek S eller M
1
2
3
35.6 CSL
Förhandsgranskar
Björn
Estimat:
4
20.7
Avancerad sök på
Analyser med hjälp
av datum intervall
5
6
Aimant de l'Equipe

18. Comment nous avons utiliser les limites d'encours

Nous avons des limites de travail en cours sur à peu près tout le tableau, écrites en rouge au sommet de chaque colonne ou d'un ensemble de colonnes. Par exemple, chaque équipe de développement essaie de limiter son encours à 5 fonctionnalités maximum à la fois. Cela signifie que si l'équipe travaille sur 5 fonctionnalités, elle ne pourra pas travailler sur une nouvelle fonctionnalité avant que l'une d'entre elles ne soit finie et déplacée dans la colonne des tests du système.



Notre limite d'encours s'applique uniquement aux fonctionnalités, ainsi les histoires techniques et les corrections d'anomalie ne sont pas incluses dans cette limite.

Les anomalies ne sont pas incluses dans la limite d'encours parce qu'elles sont souvent assez urgentes, souvent assez petites, et parce que nous n'avons pas encore trouver un moyen cohérent pour les gérer sur le tableau.

Les histoires techniques ne sont pas incluses dans la limite d'encours parce que... eh bien, laissez-moi essayer de vous expliquer...

L'objectif des limites d'encours est d'éviter le multitâche et d'empêcher la surcharge du processus aval. Par exemple, si les testeurs ont trop de travail à faire, nous ne souhaitons pas que les développeurs continuent à développer de nouvelles fonctionnalités et qu'ils leur ajoutent ainsi de la charge de travail ; ils devraient plutôt donner un coup de main aux testeurs. C'est pour cela que les limites d'encours sont utiles, elles agissent comme un signal d'alerte pour les équipes de développement.

D'un point de vue goulet d'étranglement, nos histoires techniques ont souvent l'effet opposé, à savoir qu'elles atténuent le goulet d'étranglement en aval. La plupart de nos histoires techniques concernent l'automatisation des tests et les améliorations de l'infrastructure, qui toutes les deux améliorent la qualité et facilitent la vie des testeurs.

Lorsque les tests du système deviennent un goulet d'étranglement, l'équipe essaie de terminer son cycle de tests en cours, ce qui veut dire que cela prendra quelques jours avant qu'elle puisse tirer des cartes de la colonne "prêtes pour les tests du système"

vers la colonne "tests du système en cours". La limite d'encours de 5 pour chaque équipe de développement porte à la fois sur les colonnes "développement en cours" ET "prêt pour les tests du système". Par conséquent, lorsque les tests du système deviennent un goulet d'étranglement, la limite d'encours des équipes de développement va être atteinte puisque les fonctionnalités développées sont coincées dans leur file d'attente jusqu'à ce que l'équipe tests système les tire effectivement dans sa colonne.

Que devraient donc faire les développeurs lorsque la limite d'encours est atteinte ? Ils doivent faire tout ce qui est possible pour aider ou alléger les tests du sytème. Cela veut dire des tests manuels et de la résolution d'anomalies. Mais cela peut inclure des choses plus conséquentes comme développer davantage de tests automatisés et améliorer l'infrastructure de tests. Ces choses sont considérées comme des histoires techniques. C'est pourquoi nous n'incluons pas les histoires techniques dans la limite d'encours, parce que nous souhaitons encourager les membres de l'équipe à travailler sur des histoires techniques lorsque la limite d'encours est atteinte.

Pendant un certain temps, les équipes ont essentiellement travaillé sur l'automatisation des tests, avec des cartes portant des pastilles vertes sur le tableau du projet. Ceci est un très bel exemple illustrant comment des tableaux Kanban avec des limites d'encours peuvent faciliter l'auto-organisation et l'atténuation des goulets d'étranglement.

Une autre raison pour laquelle les limites d'encours s'appliquent uniquement aux fonctionnalités est que cela reste cohérent avec nos métriques qui ne s'appliquent qu'aux fonctionnalités. C'est donc plus facile pour l'expliquer et s'en rappeler.

19. Comment nous avons récolter et utiliser les métriques du processus

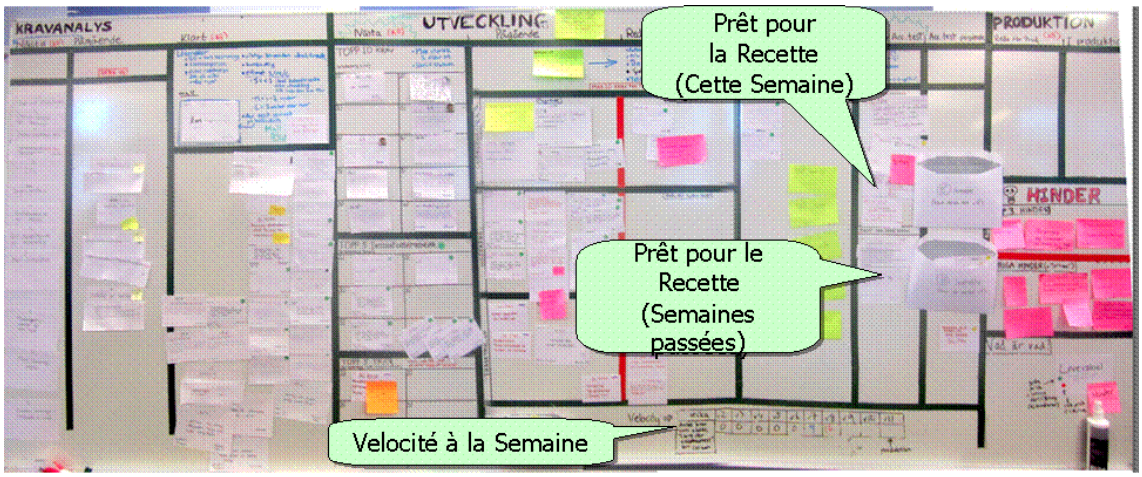
Nous collectons deux métriques pour le processus :

- la Vélocité (nombre de fonctionnalités par semaine)
- le Temps de Cycle (nombre de semaines par fonctionnalité)

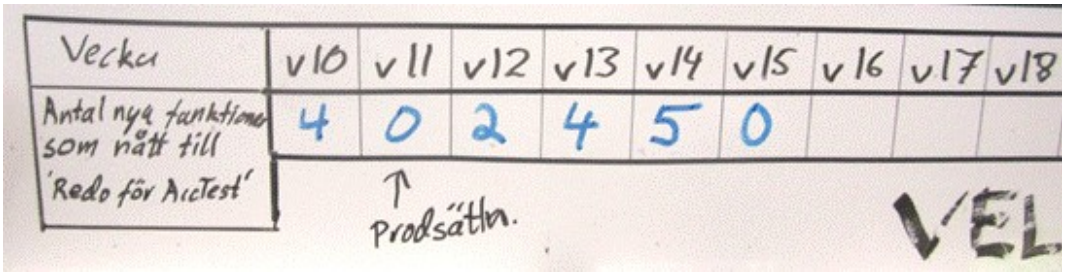
Nous le faisons manuellement, c'est étonnamment facile. Je suis surpris que tous les projets ne le fassent pas.

Vélocité (nombre de fonctionnalités par semaine)

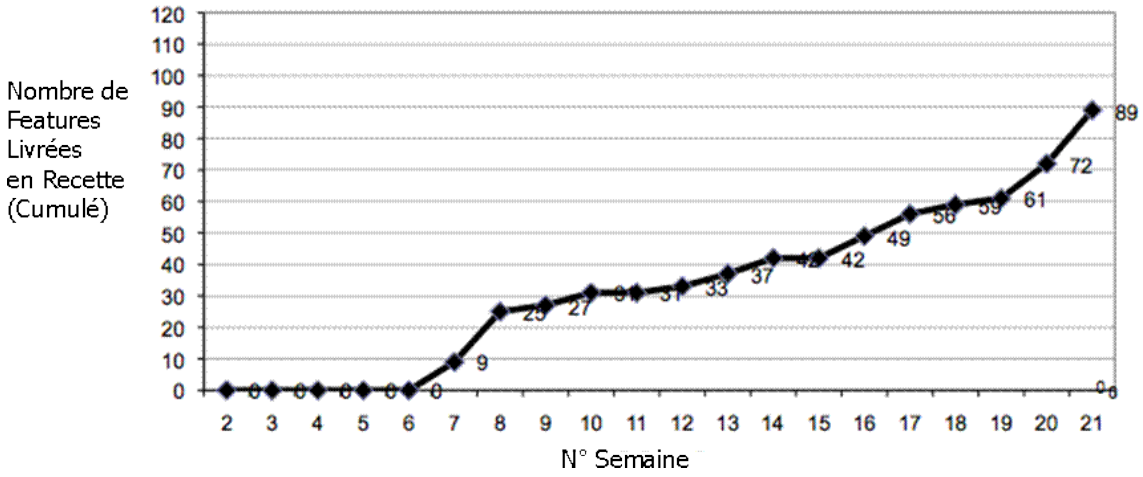
Pour la vélocité (= débit), à la fin de chaque semaine, nous comptons le nombre de fonctionnalités ayant atteint l'état "prêtes pour les tests d'acceptation (cette semaine)" dans la semaine concernée. Nous inscrivons ce nombre en bas du tableau, puis nous déplaçons ces cartes dans la colonne "prêtes pour les tests d'acceptation (semaines passées)" pour montrer qu'elles ont déjà été comptabilisées.



Voici un gros plan sur l'historique de la vélocité :



En utilisant cette information, nous générons un simple graphique burnup :

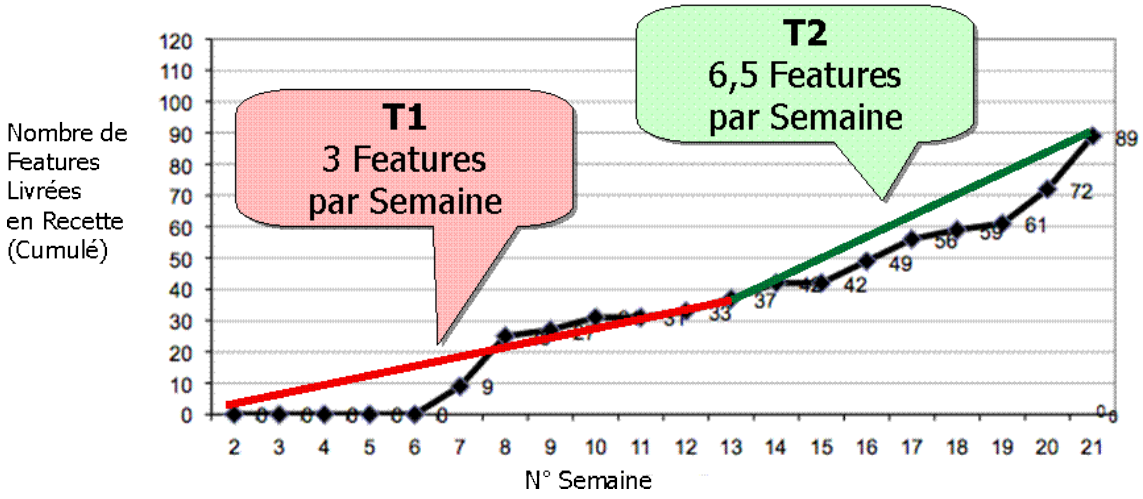


Cette information est utile à plusieurs titres.

Elle est utilisée comme "test de confiance" pour s'assurer que le plan de release est réaliste, et comme un outil pour prédire approximativement le nombre de fonctionnalités qui seront finies avant une certaine date.

Le burnup est aussi utilisé pour souligner les problèmes. Par exemple, durant les toutes premières semaines de mesure, la vélocité était de 0. Les équipes travaillaient très dur mais les tests systèmes étaient devenus un goulet d'étranglement pour différentes raisons, si bien que toutes les fonctionnalités se sont entassées en amont de l'équipe de tests. Cette situation a créé un sentiment d'urgence et les développeurs se sont donc progressivement concentrés sur le fait d'aider à passer les tests au lieu de se contenter de développer de nouvelles fonctionnalités et de les ajouter à la pile des fonctionnalités à tester.

Enfin, le burnup est utilisé pour visualiser l'amélioration du processus. Par exemple, nous pouvons voir que notre vélocité moyenne a doublé entre le T1 et le T2. Ce type de résultat bien visible permet de motiver chacun des membres de l'équipe pour continuer à améliorer le processus.



Notez néanmoins que les statistiques doivent être utilisées avec précaution. Durant les semaines ayant suivi la création de ce diagramme, la courbe s'est aplatie puisque l'équipe se concentrait sur des améliorations internes et la vélocité était à 0. Une appréciation plus réaliste de la situation dirait que la vélocité a augmenté d'environ 50% plutôt que doublé.

Nous avons décidé de mesurer la vélocité des stories techniques également, de façon à voir comment notre effort se répartissait entre les besoins client et les stories techniques. En combinant ces deux vélocités, nous allions obtenir notre capacité totale, lissant la courbe du burnup et facilitant la planification.

Pourquoi nous n'utilisons pas les points d'histoire

Rendu à ce point, vous pourriez vous demander comment nous nous en sortons en *comptant* juste les fonctionnalités. Et la taille alors ? Ne devrions-nous pas prendre en compte la taille lorsque nous mesurons la vélocité ? Supposons que toutes les fonctionnalités du T2 étaient plus petites que celles du T1, il serait donc dans ce cas parfaitement normal d'en finir deux fois plus et trompeur de déclarer que "nous avons doublé la vélocité".

En théorie, c'est correct. En pratique, pourtant, les tailles sont uniformément distribuées. Un jour, j'ai fait une petite expérience en affectant à chaque fonctionnalité un "poids" basée sur l'estimation de la taille, donc Petit = 1kg, Moyen = 2kg et Grand = 3kg. La plupart des équipes agiles appellent cela les "points d'histoire", c'est-à-dire une estimation relative de l'effort mis en oeuvre dans la fabrication de la fonctionnalité.

Voici une métaphore des plus utiles. Supposons que je porte des briques, et que je veuille mesurer la vélocité pour le faire. Ma vélocité tourne autour de 10 à 15 briques par minute. Ce n'est pas un nombre exact. Mais attendez, les briques ont des poids différents ! Qu'est-ce-que cela donnerait si nous mesurions des kg par minute, au lieu de briques par minute ? Cela nous donnerait une vélocité lissée et nous permettrait donc de prédire plus facilement le nombre de briques que chargées d'ici la fin de la journée.

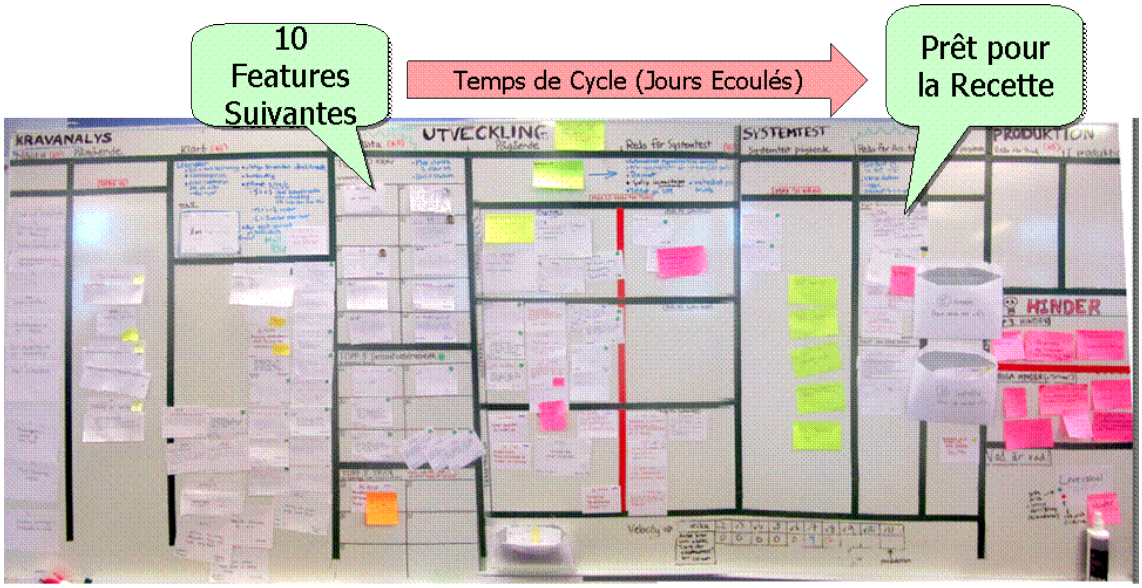
Nous avons donc mesuré chaque brique pour connaître son poids, et nous avons ensuite calculer le nombre de kg par minute que je chargeais, soit environ 20 à 30 kg par minute. Euh attendez, ce n'est pas plus précis que notre premier nombre de 10 à 15 briques par minute ! Tant que la taille des briques est en gros uniformément distribuée, ce n'est pas la peine de les peser chacune pour calculer une vélocité. Contentez-vous plutôt de compter les briques.

C'est précisément ce qui est arrivé dans notre cas. J'ai créé un burnup en kg au lieu d'un burnup en nombre de fonctionnalités, et la forme etait aussi irrégulière que précédemment. Un plus haut degré de précision ne nous a donc apporté aucune valeur ajoutée, par conséquent estimer en points de story aurait généré une perte de temps.

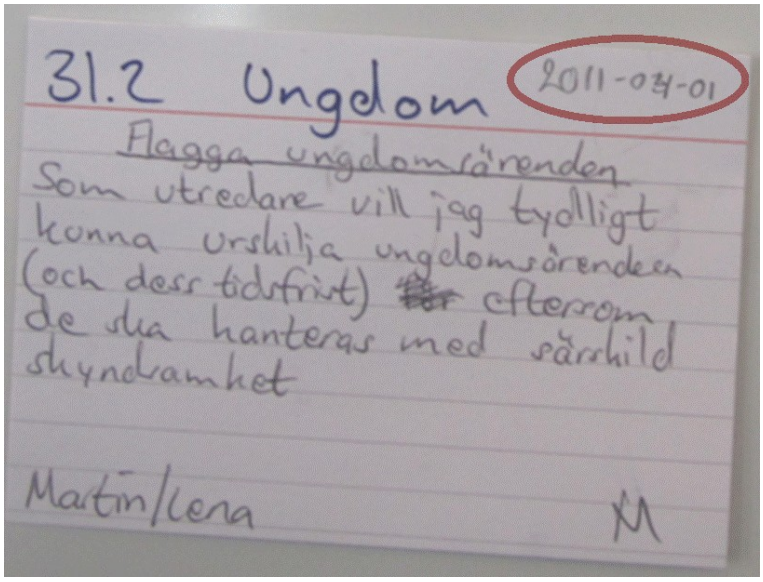
Temps de cycle (nombre de semaines par fonctionnalité)

Nous avons par contre mesuré autre chose : le temps de cycle (le temps passé dans le flux). Le temps de cycle représente le temps que cela prend de finir une chose, ce qui est dans notre cas précis "le temps que prenait une fonctionnalité X pour se déplacer de

la colonne '10 prochaines fonctionnalités' vers la colonne 'Prêtes pour les tests d'acceptation'".



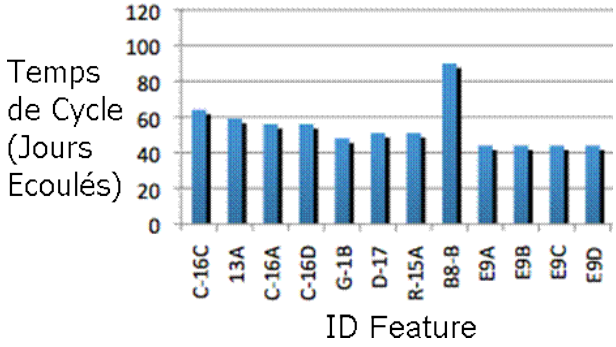
C'est également très facile à mesurer. Chaque fois que l'on choisissait une fonctionnalité pour faire partie des "10 prochaines", nous inscrivions la date de début sur la carte.



Chaque fois qu'une fonctionnalité atteint la colonne "Prêt pour les tests d'acceptation", nous inscrivons la date de fin et renseignons le nombre de jours écoulés pour cette fonctionnalité dans une feuille de calcul.

Leverabel	start	slut	Genomströmningstid (dagar)
6A	2011-02-01	2011-02-17	16
6A	2011-02-01	2011-02-17	16
26B	2011-02-03	2011-02-24	21
26A	2011-02-03	2011-02-24	21
B2.2	2011-02-08	2011-02-21	13
25.2.1	2011-02-08	2011-02-18	10
25.2.2	2011-02-08	2011-03-25	45
B8-A	2011-02-10	2011-03-09	27
T23.2	2011-02-10	2011-02-23	13
16J	2011-02-20	2011-03-28	36
A-4A	2011-03-08	2011-03-30	22
6.2	2011-03-09	2011-03-25	16

Nous le visualisons ensuite avec un histogramme, où la hauteur de chaque colonne représente le temps pris par la fonctionnalité pour traverser le tableau en partant de la colonne des 10 prochaines pour arriver dans la colonne des tests d'acceptation.

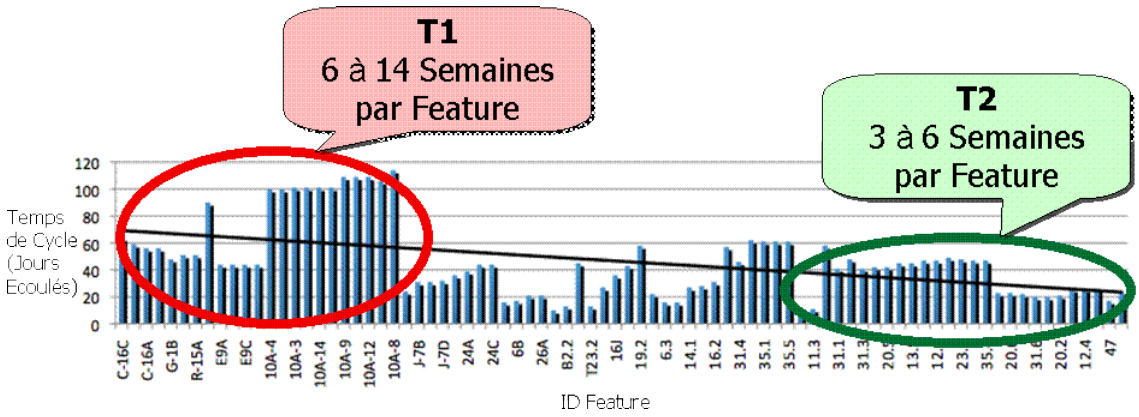


Cela se révèle très utile pour prédire combien de temps cela prendra pour finir une fonctionnalité. C'est aussi un très bon moyen de provoquer une réaction d'effroi mêlée d'horreur, étant donné que la plupart des personnes ne réalisent pas combien de temps cela prend pour traiter les choses. Cela arrive chaque fois qu'une entreprise se met à visualiser cette information :o)

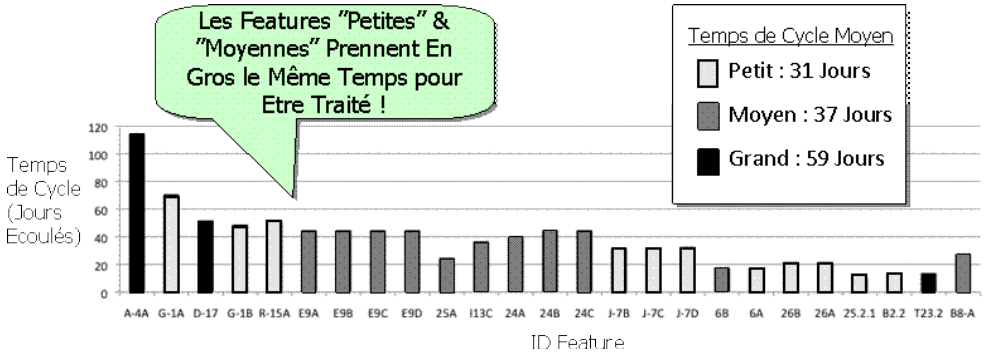
Généralement, la différence constatée entre temps écoulé et temps de travail réel est dans un rapport de 5 à 10. Cela prendra donc 20 jours calendaires pour finir une fonctionnalité qui prend réellement 3 jours de travail.

La raison pour laquelle les choses prennent généralement plus de temps que prévu, est principalement due à la parallélisation des tâches et à l'abondance de buffers et files d'attente dans lesquelles ces fonctionnalités sont coincées en attendant d'être traitées dans les étapes suivantes du processus.

La bonne nouvelle, c'est qu'une fois que vous visualisez cela et que vous commencez à vous concentrez sur le sujet, il est ensuite assez facile de raccourcir le délai de façon spectaculaire. Dans notre cas, la tendance à long terme, visualisée ci-dessous, montre comment le temps de cycle a été divisé par deux en l'espace de quelques mois.



Un autre point intéressant que nous avons noté, c'est l'absence de corrélation entre la taille des fonctionnalités et le temps de cycle. Dans le diagramme ci-dessous, un code couleur est associé à la fonctionnalité selon son estimation initiale.



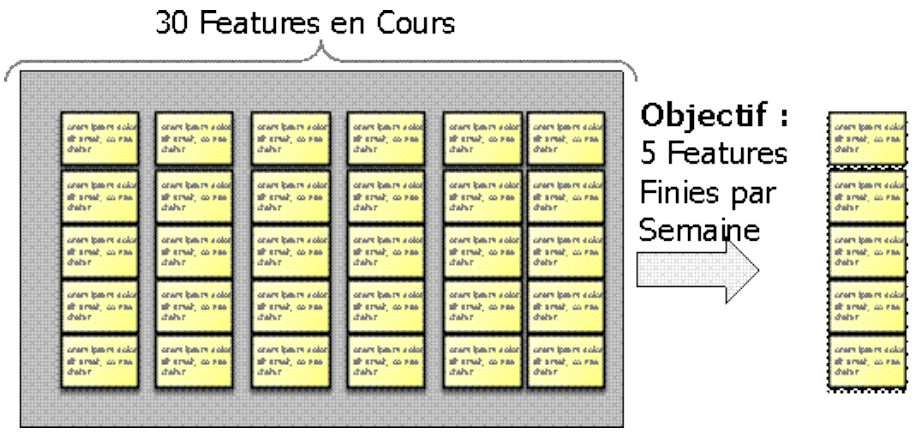
Comme vous pouvez le voir, certaines des "petites" fonctionnalités prennent jusqu'à 7 semaines, alors que de "grandes" fonctionnalités n'ont pris que 2 semaines. Il apparaît donc que la taille n'est pas le facteur principal influençant le temps de cycle ; d'autres facteurs comme le fait d'être dédié à la tâche et d'avoir accès aux ressources sont plus importants.

À un moment, nous avons traité ces informations et défini des objectifs d'amélioration. Notre but était d'avoir des objectifs ambitieux mais réalistes pour guider les efforts d'amélioration :

- Une vélocité plus stable, c'est-à-dire à peu près identique chaque semaine plutôt qu'uniformément distribuée. Cela devait nous générer moins de goulets d'étranglement, faciliter le planning de versions et lisser le flux en général.
- Une vélocité plus grande. Notre moyenne était de 3, nous avons défini un objectif de 5.
- Un temps de cycle plus faible. Notre moyenne était de 6 semaines (mais en diminution rapide), nous avons défini un objectif de 2.

Lorsque nous avons fait cela, nous avons eu une intuition intéressante. Supposons que nous avions atteint les deux premiers objectifs et obtenu une vélocité stabilisée à 5 fonctionnalités par semaine.

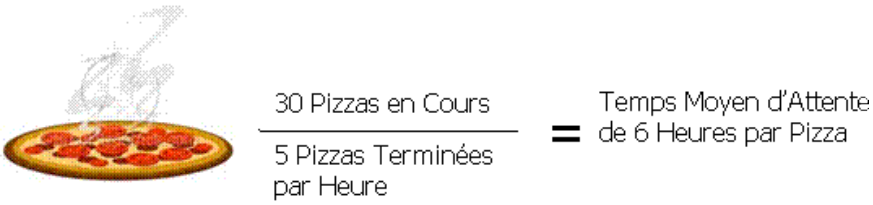
Nos informations (et photos) nous montrent que pour un jour donné, il y a généralement environ 30 fonctionnalités qui sont coincées dans les diverses colonnes files d'attente et encours du tableau du projet.



Ce qui signifie, mathématiquement, que le temps de cycle moyen sera de 6 semaines !

Supposons que nous sommes dans une pizzeria et que leur capacité est de 5 pizzas par heure. Combien de temps attendrez-vous votre pizza ?

12 minutes ?
Non, parce qu'il y a 30 autres personnes dans le restaurant qui attendent leur pizza. Dans ce cas, cela prendra 6 heures !

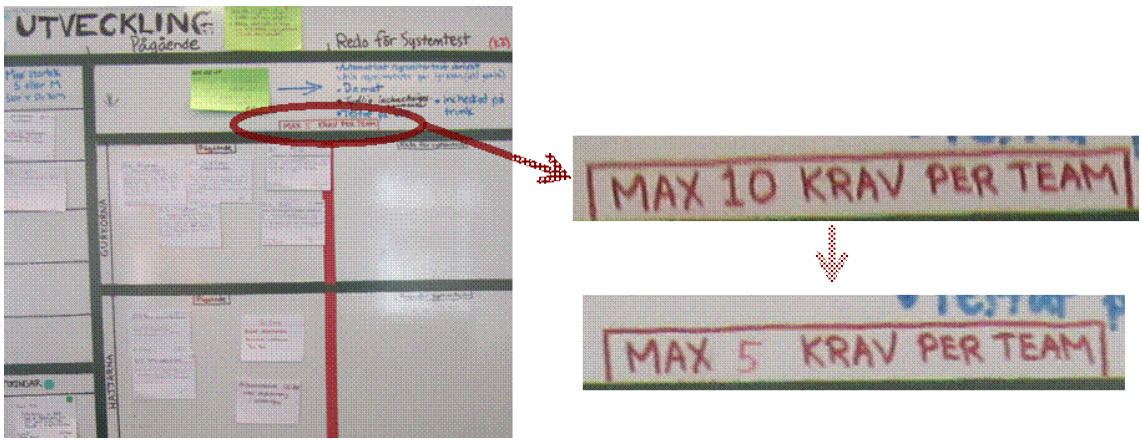


Les mêmes mathématiques s'appliquent au développement de notre fonctionnalité.

Au fait, c'est ce qu'on appelle la "Loi de Little" dans la théorie des files d'attente. On ne peut pas y échapper.

Alors, comment améliorer le temps de cycle de 6 semaines à 2 semaines ? Bien, soit en multipliant la vélocité par 3 (ce qui va coûter du temps et des efforts !), soit en réduisant l'encours par 3. À votre avis, qu'est-ce qui coûte le moins cher ? ... Exactement !

Les équipes ont donc réduit leur limite d'encours de 10 à 5 fonctionnalités par équipe.



Ce n'est pas une réduction d'un facteur de 3, mais une diminution tout de même significative. Lorsque vous réduisez l'encours, vous devez prendre en compte que si vous le réduisez trop, vous obtiendrez d'autres effets de bord, comme par exemple de nombreuses personnes inoccupées. Ce qui en retour impacte négativement la vélocité et réaugmente donc le temps de cycle. Il y a donc un équilibre à trouver. Le but est donc d'avoir une limite d'encours suffisamment basse pour maintenir la collaboration entre les membres de l'équipe et pour faire émerger les problèmes. Mais assez haute pour ne pas faire émerger tous les problèmes en même temps (ce qui génère uniquement de la frustration et un flux instable).

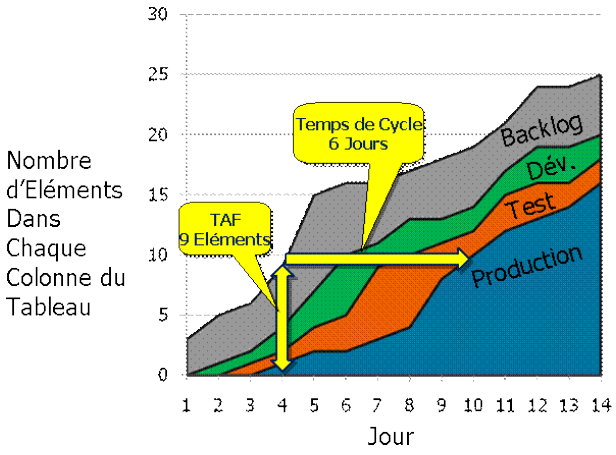
Nous n'avons cependant pas encore atteint l'objectif de 2 semaines par fonctionnalité, mais ce n'est pas très grave. Le but est de nous maintenir dans la bonne direction. Nous avons divisé le temps de cycle par deux et l'action de réduire l'encours fut l'un des points qui nous a aidés à atteindre cet objectif.

En plus, c'est sympa d'avoir des indicateurs qui montrent que nous allons dans la bonne direction.

Flux cumulés

Je n'étais pas sûr de vouloir écrire quelque chose sur ce sujet, mais bon je l'ai quand même fait. Il est courant, dans les cercles d'expertise Kanban, de visualiser la façon dont la quantité de travail évolue dans le temps. Le tableau Kanban nous montre les goulets d'étranglement en temps-réel, mais ne nous montre pas l'historique.

Voilà comment cela fonctionne. Chaque jour, comptez le nombre d'items dans chaque colonne. Visualisez le ensuite dans un DFC (Diagramme de Flux Cumulés) comme suit :

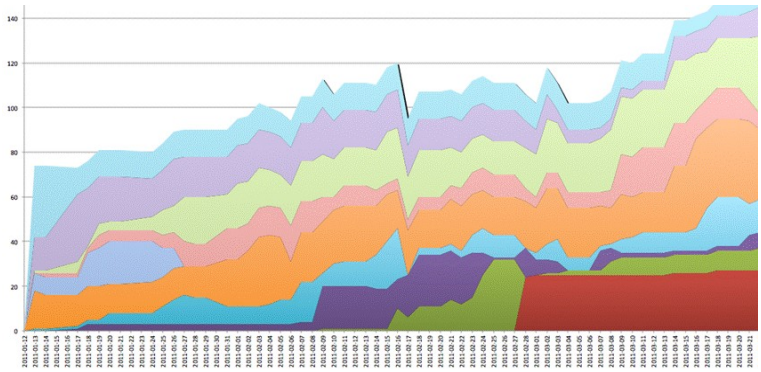


Chaque couleur représente une colonne du tableau. Chaque jour sur l'abscisse, nous montrons combien d'items sont dans chaque colonne en les empilant les uns sur les autres. Théoriquement, cela doit nous montrer où se trouvent les goulets d'étranglement, là où l'on bloque le flux, et comment le fait d'augmenter l'encours est directement corrélé à un temps de cycle plus long.

Super outil. En théorie.

"*En théorie, la théorie et la pratique sont la même chose.*
En pratique, elles ne le sont pas." - Yogi Berra

En pratique, jusqu'ici, cela n'a pas marché pour nous. Voici notre DFC :



Il est difficile d'en tirer des conclusions utiles. Et toute conclusion que nous pourrions en tirer serait probablement fausse. Par exemple, au milieu de la ligne de temps, il apparaît que nous avons soudainement supprimé du travail à réaliser, alors que ce qui est arrivé en réalité c'est que nous avons décidé d'arrêter de compter les stories techniques. Dans certaines situations, nous avons décidé de "laisser de côté" certaines fonctionnalités dans un coin du tableau, parce qu'elles étaient essentiellement en attente, et la personne qui comptait les stories du tableau chaque jour n'en tenait pas compte. Ces stories sont ensuite réapparues plus tard.

Notre DFC est apparu bien "fragile" puisqu'imprécis et trompeur à chaque fois que nous opérions des changements dans la structure du tableau ou que nous déviions du flux standard.

Cependant, nous collectons toujours consciencieusement ces données, principalement parce que j'entends les autres coachs et pratiquants Lean dire que les DFC sont utiles.

Cela prend uniquement quelques minutes par jour pour une personne, donc qui sait, nous nous en servirons peut-être un jour :o)

Efficacité du processus

C'est une des choses que nous ne mesurons pas, mais j'espère qu'un jour nous le ferons. Si notre tableau kanban était dupliqué dans un système électronique, nous pourrions définitivement la mesurer, mais cela reste aujourd'hui trop difficile à faire compte tenu de notre système kanban manuel.

$$\text{Efficacité du Cycle du Processus (\%)} = \frac{\text{Temps des Activités}}{\text{Temps Ecoulé}}$$

Le temps écoulé représente le nombre de jours que prend une fonctionnalité pour traverser le tableau (= temps de cycle).

Le temps des activités représente le nombre de jours qui ont réellement servi à traiter (ou "toucher à") la fonctionnalité. En d'autres termes, le nombre de jours pendant laquelle cette carte a séjourné dans les colonnes "en cours", à l'opposé des colonnes files d'attente/buffers.

Cela nous donne une information très intéressante telle que "hé, notre fonctionnalité X a seulement pris 2 jours pour être réalisée mais a pris en réalité 20 jours pour traverser le tableau ! Cela représente une efficacité de notre processus de 10% !".

La plupart des entreprises tournent autour de 10 à 15%, à moins qu'elles ne se consacrent à l'optimiser. Tenter d'augmenter ce nombre est un très bon moyen pour découvrir et éliminer les gaspillages.

20. Comment nous avons fait les réunions de planifications de sprints

Nos réunions de planification de sprints ne sont pas vraiment des réunions-du-manuel-du-parfait-petit-Scrum. Même s'il y a plus de similarités que de différences.

L'objectif de la réunion est de déterminer ce qui doit être fait au prochain tour, c'est-à-dire quelles fonctionnalités iront dans la colonne des « 10 prochaines fonctionnalités » ; avec Scrum, l'équipe est supposée s'engager pour un ensemble défini de fonctionnalités pour le prochain sprint, nous ne faisons pas cela. Nous n'avons même pas de sprints. Tout ce que nous souhaitons, c'est être d'accord sur le choix des fonctionnalités qui seront les prochaines à être mises en ligne, notre vélocité n'étant pas assez stable pour que nous soyons capables de prédire combien de fonctionnalités seront réalisées à court terme.

La réunion est divisée en deux parties : la préparation du backlog et la sélection du top 10.

1ère partie : la préparation du backlog

La préparation du backlog porte sur tout ce qu'il faut faire pour mettre les fonctionnalités dans l'état « prêtes pour le développement » (voir chapitre 13 « Comment nous avons défini les notions de prêt et de fini »).

Nous avions l'habitude de procéder ainsi durant la première moitié de la réunion de planification de sprint. L'équipe d'analystes métier présente les prochaines

fonctionnalités à développer (oui, cette équipe occupe le rôle Scrum de Product Owner, essentiellement parce que ses membres sont les plus proches des clients et des utilisateurs). Nous séparons le groupe en petites unités pluridisciplinaire : généralement un analyste métier, un développeur et un testeur. Ces derniers estiment les fonctionnalités en utilisant le planning poker (consulter http://www.crisp.se/planningpoker pour plus d'informations sur ce sujet) et écrivent P, M ou G sur la carte. Si la carte est G, alors ils la découpent davantage, ou décident de l'écarter des « 10 prochaines fonctionnalités » pour le reste de la séance. Ils discutent et se mettent d'accord sur les tests d'acceptation appropriés et les écrivent au verso.

Après 20 à 30 minutes, nous avons une jolie pile de cartes avec des fonctionnalités bien préparées.

2e partie : la sélection du top 10

C'est là où nous examinons la pile des fonctionnaliés à disposition et discutons lesquelles devraient être dans le top 10. Généralement, la liste du top 10 n'est pas vide avant de commencer, 2 ou 3 fonctionnalités peuvent déjà être présentes. Dans ce cas, nous évaluons ces fonctionnalités par rapport aux nouvelles. Le thème de la conversation est « faire abstraction du monde qui nous entoure pour se concentrer sur le contenu du prochain sprint, quelles seront les 10 fonctionnalités du top 10 ? ».

Un certain nombre de points influencent notre décision :

- **Valeur métier** : quelles fonctionnalités seraient les plus appréciées par les clients ?
- **Connaissance** : quelles fonctionnalités apporteront de la connaissance (et par conséquent réduiront les risques) ?
- **Utilisation des ressources** : nous voulons équilibrer les domaines fonctionnels afin que chaque équipe ait du travail
- **Dépendances** : quelles fonctionnalités sont de préférence à construire les unes avec les autres
- **Testabilité** : quelles fonctionnalités sont logiquement à tester les unes avec les autres et devraient être développées ensemble ?

Préparer le backlog en dehors de la réunion de planification du sprint

Après avoir fait quelques réunions de planification de sprint, nous avons remarqué que la préparation du backlog prenait un temps assez long, et quelquefois la réunion de planification de sprint se terminait au pas de course (car nous souhaitons dérouler la réunion en temps limité). Maintenant la plus grande partie de la préparation est faite séparément, avant la réunion de planification de sprint. Généralement, un des analystes métier aura une conversation avec un développeur et un testeur pour discuter d'une fonctionnalité à venir et le résultat de la conversation peut être une décomposition, une estimation et un test d'acceptation.

Quelques réaménagements du backlog peuvent encore être faits pendant la réunion de planification de sprint, mais l'un dans l'autre, il est mieux de préparer le plus possible à l'avance la réunion de planification de sprint. Je constate cette tendance dans la plupart des organisations avec lesquelles j'ai travaillé.

21. Comment nous avons fait le planning de release

Nous connaissions notre vélocité, elle était habituellement de 3 fonctionnalités par semaine en moyenne et aujourd'hui elle est de 4 à 5. C'est une information utile à long terme pour un planning de release. Nous ne savons pas exactement quelle sera notre vélocité dans le futur, mais il est raisonnable de penser qu'elle sera probablement de 3 à 5 fonctionnalités par semaine.

Ainsi, si quelqu'un demande « Combien de temps faudra-t-il pour implémenter ces nouveaux éléments ? » (en déroulant une liste de fonctionnalités) ou « Parmi ces nouveaux éléments, combien pourrons-nous en faire d'ici Noël ? », nous pouvons donner une réponse réaliste si nous connaissons le nombre de fonctionnalités.

Le problème est que, pour un planning à long terme, nous ne connaissons pas le nombre de fonctionnalités. Tout ce que nous avons est un tas d'idées plutôt vagues. Nous pourrions les appeler des épopées, ou des domaines de fonctionnalités. Certaines peuvent être vraiment vraiment très grandes.

OK, j'ai mentionné précédemment que nous n'estimions pas les fonctionnalités en points d'histoire car il s'avère que les tailles estimées sont distribuées de manière égale, par conséquent les points d'histoire n'apportent pas de plus-value. Cependant pour des plannings à long terme cette logique ne tient plus debout puisque nous considérons plus des épopées que des fonctionnalités. Même si notre vélocité est de 3 à 5 fonctionnalités par semaine comprenant parfois une épopée, cela ne signifie pas 3 à 6 épopées par semaine !

La solution est simple. Prendre chaque épopée et estimer en combien de fonctionnalités elle sera découpée. Cette estimation requiert beaucoup d'effort de la part des analystes métier, des développeurs et des testeurs. Le processus est, en fait, similaire à l'estimation en points d'histoire, mais nous utilisons le terme « fonctionnalité » plutôt que « point d'histoire », et posons la question suivante : « Alors, combien de fonctionnalités pour cette épopée ? ».

Une fois que nous avons l'information, nous pouvons compter le nombre total de fonctionnalités et le divisons par 3 à 5 fonctionnalités par semaine. Cela nous donne une réponse du genre : « Nous pouvons probablement construire tous ces nouveaux éléments d'ici 6 à 12 mois ». Une estimation grossière, mais basée sur des données réelles.

Lorsque la vélocité se stabilisera, nous serons capables de faire de meilleures prédictions, ainsi notre réponse pourrait devenir 8 à 10 mois (aussi longtemps que nous ne changeons pas trop la taille de l'équipe).

22. Comment nous avons fait la gestion de configuration

Ce sera un court chapitre. Non pas parce que c'est un sujet inintéressant, ni parce que la gestion de configuration est simple. Simplement parce que je commence à fatiguer et que je voudrais terminer ce livre :o)

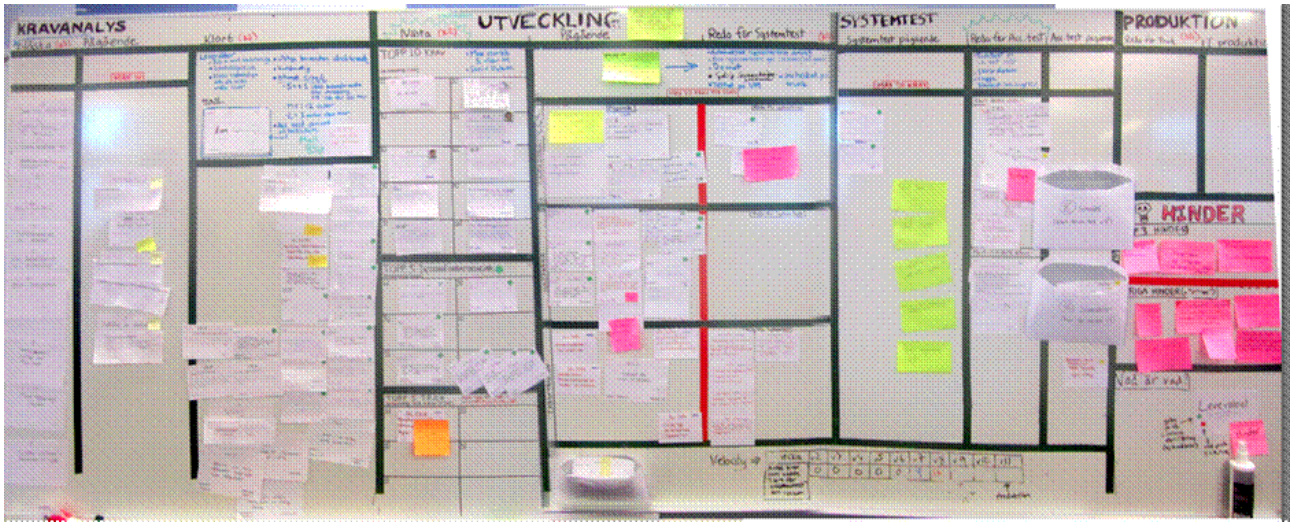
Nous avions beaucoup de défis à relever car nous avons développé assez rapidement un système complexe avec plusieurs équipes. Une leçon que nous avons apprise est que nous aurions dû avoir notre système de gestion de configuration en état avant de passer de 30 à 60 personnes. Pendant une période relativement longue, nous avons gravement endommagé des troncs et des branches dans notre système de gestion de configuration.

Finalement, nous avons décidé d'implémenter un « modèle principal », un schéma à tronc stable décrit dans mon article « Gestion de configuration avec plusieurs équipes ».

Le changement a été mouvementé, mais cela nous a certainement aidé à mettre les choses en ordre. Je n'écrirai pas plus ici sur ce sujet, parce que je finirais probablement par dupliquer l'article précédemment cité. Lisez-le si vous êtes intéressé par ce genre de choses. Ou mieux encore, lisez « Livraison continue » par Jez Humble et David Farley. Un gros pavé, mais avec plus de choses à se mettre sous la dent.

23. Pourquoi nous avons utilisé seulement des tableaux kanban physiques

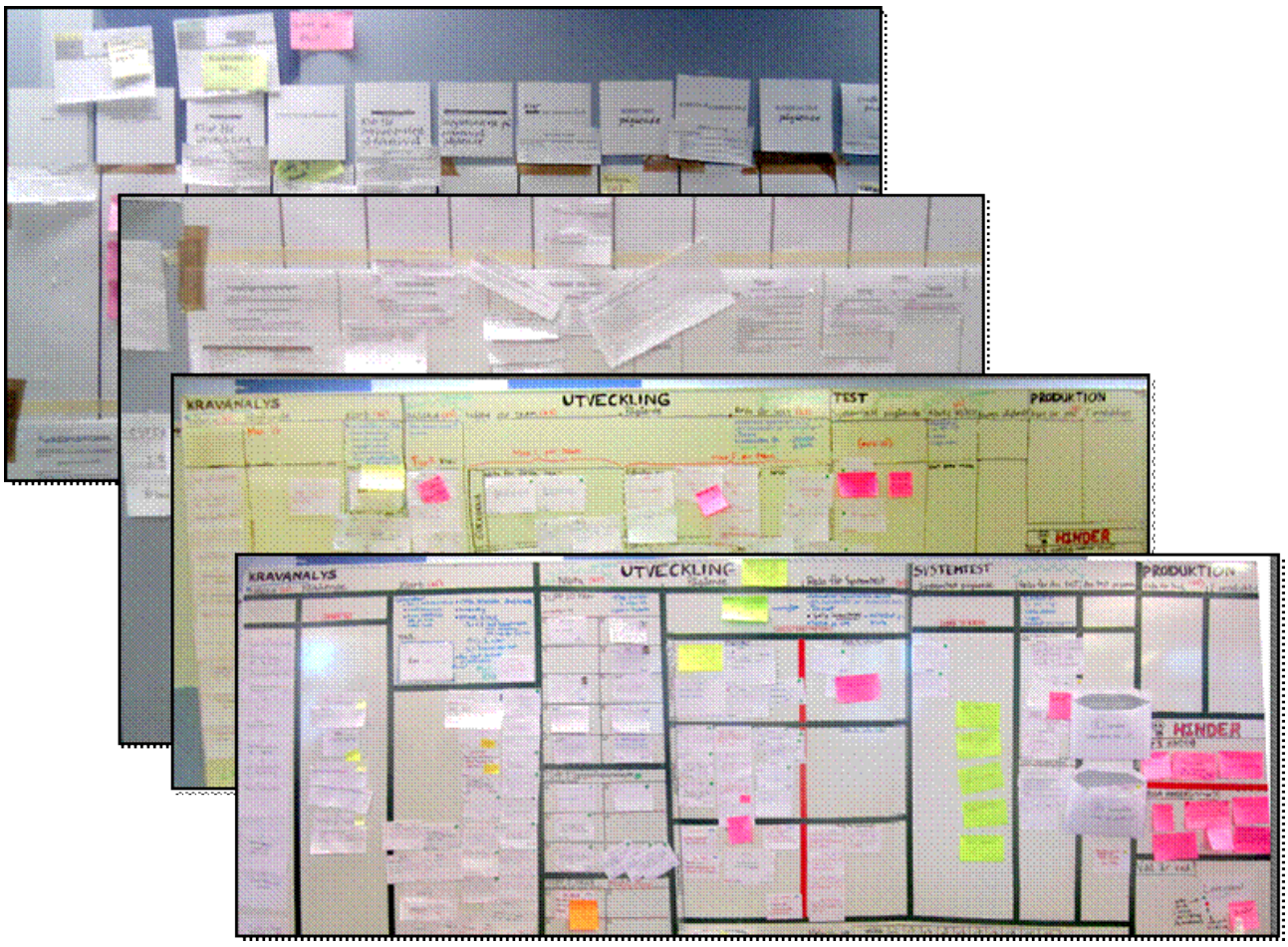
Nous possédons des systèmes de suivi électronique pour compléter les tableaux physiques kanban, comme un gestionnaire d'anomalies, différentes feuilles de calculs pour les métriques et les plannings de livraisons. Le tableau projet étant le « tableau de référence», nous nous sommes mis tous d'accord pour que ce tableau reflète toujours la vérité et qu'il soit mis à jour en temps réel.



Tout autre document électronique dupliquant des informations de ce tableau est considéré comme un «reflet» de ce tableau, si des choses sont désynchronisées, ce tableau est toujours la référence.

Alors pourquoi utilisons-nous ce machin analogique moche et désordonné utilisant du ruban adhésif, des notes adhésives et du texte manuscrit, lorsqu'il existe plein d'outils électroniques alléchants ? La plupart des outils peuvent générer automatiquement toutes sortes de statistiques détaillées, peuvent sauvegarder, peuvent être accessibles depuis l'extérieur, peuvent offrir différentes vues, etc.

Eh bien, une raison est sa facilité à évoluer.



Notre tableau a changé de structures plusieurs fois. Il a pris quelques mois avant de commencer à se stabiliser. C'est alors que nous avons commencé à utiliser du ruban adhésif noir pour délimiter les colonnes, avant cela nous avions l'habitude de tracer les lignes à la main parce qu'elles changeaient très souvent. Mais nous pouvons encore déplacer le ruban adhésif si nous en avons besoin.

Voici quelques exemples des changements qui se sont produits :

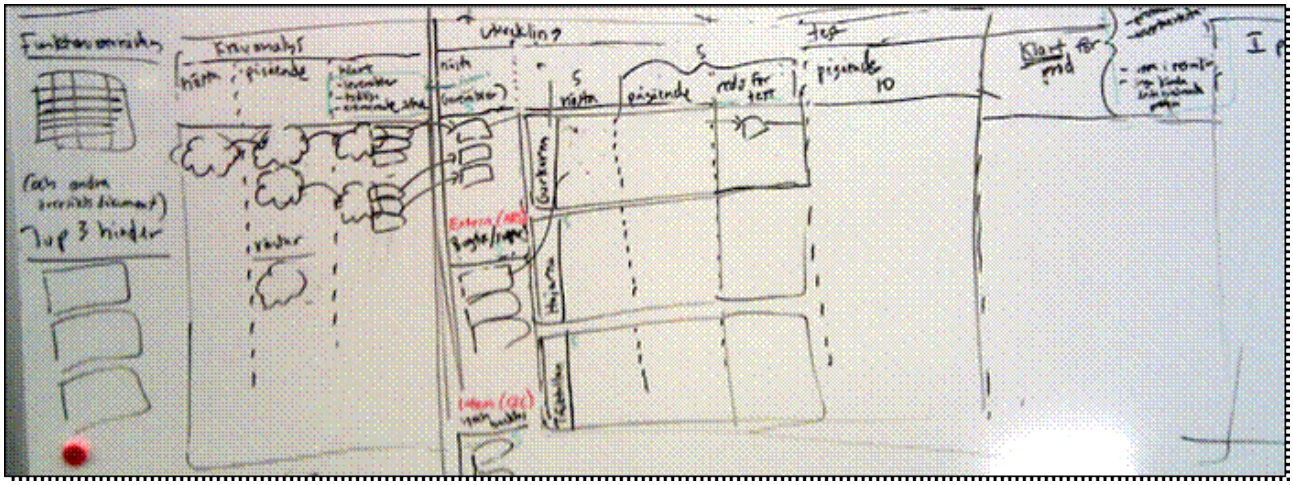
- Ajouter ou supprimer des colonnes
- Ajouter ou supprimer des couloirs horizontaux, à l'intérieur d'une colonne, ou à travers tout le tableau
- Ajouter un nouveau type d'élément sur le tableau (fiche cartonnée, note adhésive, aimant, ruban adhésif coloré).
- Écrire les règles telles que la « définition du fini »
- Écrire les métriques telles que la vélocité
- Ajouter des signalétiques de couleur telles que « un texte écrit en rouge est une anomalie » ou « une étiquette rose signifie un obstacle »
- Utiliser des enveloppes pour regrouper toutes les fonctionnalités livrées ensemble, et écrire la version de la livraison dessus
- Permettre que certains éléments soient placés sur la frontière entre deux équipes parce qu'elles se les partagent
- Diviser une fonctionnalité en plusieurs sous-fonctionnalités, et les garder ensemble en écrivant un mot-clé en haut de chaque sous-fonctionnalité.

Tous ces changements sont triviaux à implémenter. N'importe qui, même un enfant de 5

ans, pourrait réaliser n'importe lequel de ces changements physiquement sur le tableau, une fois que nous savons exactement à quoi nous voulons qu'il ressemble.

Je n'ai pas encore vu d'outils électroniques qui puissent faire tout ce qu'il y au-dessus – à l'exception peut-être d'un programme de dessin générique comme Google Drawing (qui nous utilisons pour notre tableau kanban distribué à Crisp, mais c'est une autre histoire). Et si nous ajoutons la règle suivante : n'importe qui devrait pouvoir réaliser le changement en moins de 15 minutes sans entraînement, eh bien, dans ce cas, un mur physique de cartes est dur à battre.

À un moment, nous avons refait la totalité du tableau en nous basant sur ce dessin :



Cela a pris à peu près une heure à créer le vrai tableau basé sur cette esquisse. Encore une fois, la plupart des outils électroniques que j'ai vus ne peuvent pas faire cela. Et ceux qui le pourraient requièrent une connaissance d'un niveau expert.

Le seconde raison pour laquelle nous utilisons un tableau physique est la collaboration.

Le « cocktail quotidien » (cf. chapitre 7) serait très difficile à faire sans tableaux physiques.

Si nous avions un tableau kanban électronique, nous pourrions utiliser un projecteur pour l'afficher sur le mur. Mais nous perdrions la plupart des interactions, à ce moment où les gens prennent une carte du mur et l'agitent en en parlant, ou lorsqu'ils écrivent des trucs sur les cartes ou les déplacent pendant la réunion. Les gens apprécieraient de mettre à jour le tableau en étant assis à leur poste de travail, c'est plus pratique mais moins collaboratif.

Un schéma clair que j'ai remarqué avec tous mes clients est que les tableaux comme celui-ci peuvent vraiment changer la culture d'une organisation, et c'est ce qui s'est certainement produit avec le projet PUST.

Durant une rétrospective du projet PUST, un des premiers points qui est venu sous « continuer à faire » a été « continuer à utiliser les tableaux kanban pour visualiser le travail ».

Nous avons discuté de l'introduction d'un outil électronique pour dupliquer les parties du tableau à un plus haut niveau, de cette manière, nous pourrions automatiser certaines des métriques et faire un tableau électronique haut niveau visible des dirigeants et des autres parties prenantes qui ne sont pas dans le même bâtiment. Cela coûterait quand même quelques efforts pour synchroniser les tableaux physiques et numériques, mais cela peut valoir le coup. Cependant, ce ne serait qu'un complément au tableau physique, pas un remplacement.

24. Glossaire pour démystifier certains termes

Toutes les personnes qui étaient sur le projet PUST parlaient suédois. La plupart des termes du jargon Lean & Agile semblaient étranges pour des personnes normales, en particulier les non-anglophones. Des termes comme "Backlog produit", "Rétrospective", "Histoires utilisateurs", "Vélocité", "Scrum Master" et "Points d'histoires" risquaient donc de laisser sur le bord de la route les non-techniciens.

J'ai donc essayé de démystifier certains termes autant que possible sur ce projet. Il n'eût pas été opportun de perdre des personnes en cours de route. Le fait de soigneusement choisir les termes à employer s'est révélé très utile, laissez-moi donc partager notre glossaire avec vous.

Inutile de dire que ce chapitre est sutout pertinent pour les lecteurs sudéois :o)

Notre terme	**Traduction littérale en français**	**Ce que nous voulions dire (termes techniques associés/synonymes)**
Processförbättringsmöte	Réunion d'amélioration du processus	Rétrospective du sprint
Leverabel	Livrable	Fonctionnalité, élément du backlog produit
Kundleverabel	Livrable client	Histoire utilisateur (à opposer aux histoires techniques et autres choses qui ne sont pas orientées client)
Funktionsområde	Domaine fonctionnel	Epopée, Thème
Teamledare	Chef d'équipe	Scrum Master
Projekttavla	Tableau du projet	Tableau Kanban au niveau du projet
Teamtavla	Tableau de l'équipe	Tableau hybride Scrum / Kanban au niveau de l'équipe

Le terme "Leverabel" se révéla très utile. Auparavant, le terme "krav" (= "exigences") se référait à tout et n'importe quoi. Aujourd'hui, il y a une distinction claire entre "leverabel" et "funktionsområde".

25. Conclusion

Fichtre. Ce livre n'a pas cessé de grossir et grossir. Je n'ai pas réalisé à quel point nous avions appris des choses.

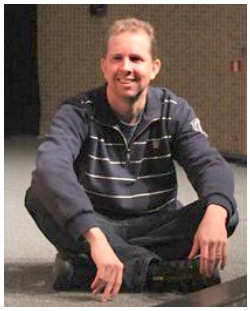
Dans tous les cas, j'espère que cela vous a été utile !

Qu'avez-vous appris à la lecture de ce livre ? Que mettrez-vous en oeuvre ? Avez-vous des expériences similaires à partager ? N'hésitez pas à m'envoyer tout type de feedback à henrik.kniberg@crisp.se

Si vous avez apprécié ce livre et que vous en voulez encore plus, je vous invite à garder un oeil sur mon blog http://blog.crisp.se/henrikkniberg

Bonne journée !

Henrik, le 10/06/2011



PS : si vous souhaitez en savoir un peu plus sur moi, consultez ma page web :
http://www.crisp.se/henrik.kniberg

PS : si vous souhaitez de l'aide pour améliorer votre processus de développement, consultez : http://www.crisp.se (principalement en suédois) ou contactez-nous à info@crisp.se. Nous sommes un groupe de coachs et de développeurs ayant une attitude agile.

Mise à jour (Décembre 2011):
Le livre, qui constitue une notable amélioration de la version draft, est désormais disponible ! Vous pouvez l'acheter ici si vous souhaitez soutenir l'auteur :
http://pragprog.com/book/hklean/lean-from-the-trenches

The Pragmatic Programmers

Lean from the Trenches

Managing Large-Scale Projects with Kanban

Henrik Kniberg

Foreword by Kent Beck

Edited by Kay Keppler

26. Biographie des traducteurs

Claude Aubry
L'agilité je suis tombé dedans quand j'étais petit. En fait pas tout à fait, mais ça fait si longtemps que je ne me souviens plus de ma vie d'avant. Comme j'aime le rugby, j'ai aimé Scrum, qui me l'a bien rendu en me procurant du plaisir - et des affaires. Dès 2005, j'ai dégusté du Scrum glacé en cornet, à partir de 2006 j'ai commencé à raconter mes aventures agiles dans le blog Scrum, Agilité & rock'n roll et en 2010 j'ai carrément publié Scrum le guide pratique de la méthode agile la plus populaire. Aujourd'hui, comme c'est le printemps, je gazouille dès le matin (@claudeaubry).

Sylvain Fraïssé
Développeur depuis pas mal de temps, je suis rentré dans l'agilité par la porte des techniques d'ingénierie poussées pour la qualité des développements et pour ses valeurs humaines. Aujourd'hui, je ne vois pas comment je pourrais travailler dans une équipe qui n'en adopterait pas les valeurs. Dans une démarche d'amélioration continue, je lis régulièrement des articles, et peut être amené à relire des traductions de mes camarades ;-)

Nicolas Mereaux
Pendant des années et encore maintenant, je me suis posé les questions « comment ça marche ça ? », « et si on change le truc là, ça fait quoi ? », puis un jour je suis tombé sur l'agilité (aïe) ou plutôt elle m'est tombée dessus (ouille) depuis j'ai rajouté les questions « pourquoi devons-nous toujours faire comme ça, si ça ne fonctionne pas ou alors si mal que personne n'est content ? », et par conséquent « et pourquoi pas, alors pour de vrai, faire de l'agilité pour faire avancer les choses en respectant les personnes qui le font ? ». Depuis 2008, c'est ce que j'essaye de faire et de montrer aux autres ; je ne peux plus m'en passer !

Antoine Vernois
J'ai un doctorat en Informatique de l'ENS Lyon. Je pratique intuitivement les grands concepts de l'agilité pendant plusieurs années avant d'en découvrir leurs formalisations en 2008 et devient alors ScrumMaster puis Coach Agile. Je m'implique activement dans les communautés ayant trait à l'agilité et au développement logiciel : membre de la SigmaT, du Scrum User Group France, du Java User Group Toulouse et, début 2012, je suis à l'origine de la communauté Software Craftsmanship à Toulouse. Je vis actuellement sur Toulouse, mais j'apprécie de me déplacer un peu partout. Vous pouvez me suivre sur mon blog (http://blog.crafting-labs.fr) où je parle d'agilité et de développement logiciel et sur Twitter (@avernois).

Fabrice Aimetti
L'Agilité, je ne suis pas tombé dedans tout petit. Mais du jour où je l'ai rencontrée (début 2009), je ne l'ai plus quittée. J'en vis maintenant à plein temps depuis la création de la marque Agilarium® (début 2012). Je blogue agile, je tweete agile, je traduis agile et je joue agile. Vous aussi, vous pouvez le faire !