L'apprentissage discipliné

le successeur de la gestion de risques

Alistair Cockburn, Humans and Technology

En résumé – L'apprentissage discipliné aussi connu sous l'expression « apprendre au plus tôt, apprendre souvent » dépoussière le développement agile dit « naïf », et la gestion traditionnelle des risques. Il remplace de manière avantageuse cette phrase épouvantable « échouer au plus tôt, échouer souvent ». L'apprentissage discipliné, déjà présent dans plusieurs domaines d'excellence, est une démarche riche, créative et gratifiante.

Introduction

Le développent agile dit « naïf », de part sa simplicité, fonctionne remarquablement bien. Toutefois, il s'avère insuffisant et loin d'être optimal dans certaines situations. L'apprentissage discipliné vient compléter l'agile sur différents points.

La gestion traditionnelle des risques répond généralement à la question : comment éviter l'échec - plutôt que - comment arriver à livrer avec succès. L'apprentissage discipliné modernise la gestion de risque en y incorporant certains principes issus du développement agile.

L'apprentissage discipliné n'est ni évident à prendre en main ni pour les mauviettes. Il est très fréquemment utilisé dans différents domaines par des équipes ayant un bon niveau, et permet à celles-ci de livrer avec succès en dépit de circonstances difficiles.

Prenons donc comme base de référence, la méthode de travail, encore répandue, selon laquelle une intégration ou une livraison majeure a lieu à la fin d'une longue période de travail sans intégration ou livraison (cf. Illustration 1). Ce mode d'intégration ou de livraison dans un projet n'est pas forcément représentatif uniquement pour un cycle en cascade, cette courbe n'est donc pas liée à ce type de cycle . Elle décrit simplement une stratégie rencontrée un peu partout.

L'illustration 1 montre sur l'axe des abscisses - le temps, sur la ligne en pointillée - le coût du projet qui augmente constamment avec le temps, sur la ligne continue – l'augmentation du niveau d'apprentissage pendant que les équipes projet travaillent, discutent, conçoivent mais cette augmentation est insuffisante pour que cet apprentissage (et les surprises éventuelles) ne donne ses fruits tout de suite après le moment de l'intégration ou de la livraison.

Sur cette illustration, la connaissance arrive bien tard dans la vie du projet et cela bien après que les coûts aient atteint un certain niveau.

Ce que nous recherchons c'est comment pouvons-nous apprendre plus tôt dans la vie du projet à quel moment les changements peuvent être faits à bas coût. Et c'est là qu'intervient la créativité et la discipline.

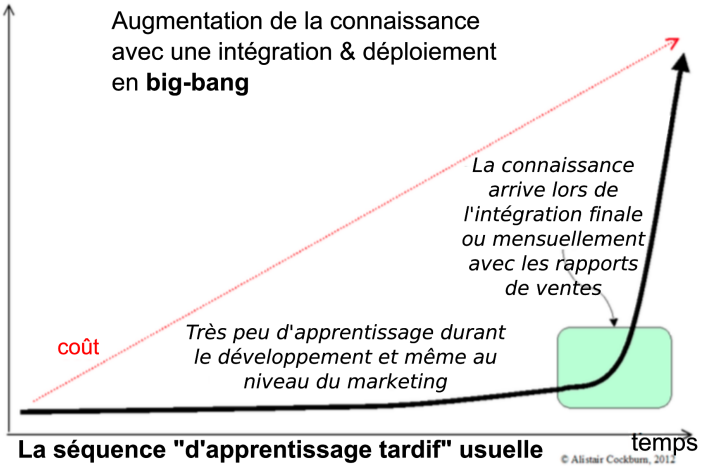

Illustration 1. La stratégie « d'apprentissage tardif » usuelle.

Quatre sujets d'apprentissage

L'équipe a (au minima) l'occasion d'apprendre dans quatre catégories :

- À propos de ce qu'elle devrait vraiment réaliser, sans prendre en compte ce qui devrait être fait selon elle dès le début
- À propos de savoir si elle a les bonnes personnes dans l'équipe, et pour ces personnes, comment travailler ensemble le mieux possible
- À quel niveau leurs choix techniques se sont avérés faux
- Combien cela va-t-il coûter à développer

Dans la stratégie figurant dans l'Illustration 1, tout cela s'apprend tard dans la vie du projet, à peu près au moment où les différentes parties du système sont intégrées et déployées, lorsque les clients donnent enfin leurs avis et leurs retours d'informations sur le résultat. Cette connaissance arrive bien trop tard pour vraiment bénéficier au produit.

L'approche de l'apprentissage discipliné consiste à appliquer cette même courbe d'apprentissage « brisée » mais à très petites doses, très souvent et délibérément, pour que chaque étape donne des informations qui puissent être utilisées et ajuster ainsi les quatre catégories d'apprentissage. Les gains obtenus ne sont pas uniquement liés à la réduction des risques pour la dernière livraison, mais aussi à la capacité pour les sponsors de la piloter très finement, à la fois en terme de date de livraison, de fonctionnalités livrées et de qualité.

L'illustration 2 montre l'approche de l'apprentissage discipliné. Les quatre sections suivantes du présent article décrivent différentes stratégies pour chacune des quatre catégories.

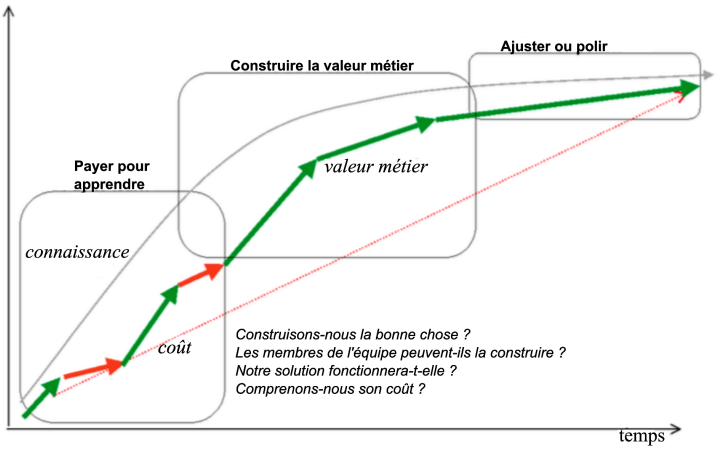

Illustration 2 Appliquer le principe: apprendre au plus tôt, apprendre souvent.

Apprendre ce qui devrait être réalisé

La question la plus importante et la plus difficile est : est-ce que les gens vont aimer, acheter et utiliser ce que nous sommes en train de réaliser ?

La réponse à cette question arrive généralement trop tard. Toutefois, récemment, des stratégies ont vu le jour qui permettent d'accélérer ce processus d'apprentissage. Les stratégies à appliquer sont relativement simples, mais exigent discipline, patience et savoir faire preuve de bonne volonté pour orienter sa course selon les résultats obtenus.

Voici quelques exemples de stratégies :

- Utiliser du prototypage papier
- Avoir des utilisateurs ambassadeurs
- Livrer au plus tôt
- Faire des livraisons à vide ou manuelle

L'utilisation de prototypage papier [1] et d'autres stratégies du même type proviennent des communautés du design centré utilisateur [2] et implique rien qui ne soit plus compliqué que de mettre une maquette dans la main des utilisateurs qui pourront alors réagir très tôt à ces éléments de conceptions. Élaboré à bas coût, très tôt dans le cycle de développement, ces prototypes permettent à l'équipe de développement de changer d'avis quant à sa manière de procéder.

Un « utilisateur ambassadeur » est un utilisateur bienveillant à qui l'équipe va livrer un produit incomplet mais en cours de développement. Cet utilisateur cassera sans doute le système en quelques minutes, mais il donnera des retours intéressants de son point de vue (même limité). La différence entre l' « utilisateur ambassadeur » et le « prototypage papier » est que l'ambassadeur fait face au vrai système en cours de développement, et non à une maquette du système.

Une « livraison au plus tôt » consiste en un déploiement complet du système avec des fonctionnalités réduites. L'intention derrière cela est, avant tout, d'apprendre ce qui s'avère être incorrect par rapport au produit tel qu'il a pu être envisagé initialement, mais aussi potentiellement et de manière plus significative, comment la seule présence du produit peut changer le regard et les réflexions sur ce qui aurait pu être fait dès le début. La « livraison au plus tôt » reconnaît le fait qu'une fois que les personnes ont commencé à utiliser un système, leurs habitudes et leurs besoins changent, de manière souvent imprévisibles. Livrer une version réduite du système permet à l'équipe de développement de rassembler de nouvelles informations et d'ajuster les priorités sur ce qui devrait être développé.

Toutes les techniques ci-dessus ne sont pas nouvelles bien qu'elles soient fréquemment ignorées, elles sont déjà présentes dans la littérature agile grand public.

Deux des plus intéressantes stratégies ayant émergées ces dix dernières années qui proviennent, qui sont utilisées et documentées par la communauté *lean startup* sont : la livraison à vide et la livraison manuelle (c'est ainsi que je les nomme).

La « livraison à vide » [3] est particulièrement bien adaptée pour les produits en ligne. Initialement, la seule chose qui est faite, c'est de détecter si quelqu'un a cliqué sur un lien ou a accédé à une fonctionnalité. Il n'existe aucune implémentation derrière le clic en façade. En mesurant ces clics, l'équipe peut réduire ou séquencer les fonctionnalités développées pour se concentrer d'abord sur celles qui ont attiré le plus d'attention. Le système évolue dans la direction où cela tire le plus.

La « livraison manuelle » est décrite dans le livre de Eric Ries, Lean Startup [4]. Dans cette stratégie, l'équipe dépense ce qu'il peut sembler une somme astronomique d'argent pour livrer manuellement les produits, pour la simple et bonne raison que les procédures manuelles peuvent être mises en place et changées à très faibles coûts. En livrant manuellement, l'équipe peut changer ce qu'offre le produit après chaque achat, le faisant évoluer sur la base de ce que le client indique souhaiter vraiment.

Ajuster les décisions de conception

Les erreurs de conception proviennent :

- De choix de technologies qui ne fonctionnent pas comme promis
- D'une non-communication entre les gens, ce qui a comme résultat des hypothèses fausses sur le travail de chacun
- D'erreurs et d'omissions inévitables dans la conception

Ces erreurs sont découvertes et corrigées en utilisant les stratégies suivantes :

- Le squelette ambulant
- Le développement micro-incrémentales
- Les pitons (*spikes* pour le terme anglo-saxon – NdT)
- Le fractionnement des *user stories*

La stratégie du « squelette ambulant » [5] consiste de la part de l'équipe de connecter les éléments de l'architecture de manière très ténue. En créant ce système simple mais complet, l'équipe est en capacité de découvrir en avant-première les surprises qui pourraient survenir avec les technologies utilisées dans le produit.

Une fois que le système est connecté de manière ténue, les équipes responsables de l'infrastructure et des fonctionnalités ajoutent chacune leur partie dans le système. Il est plutôt inhabituel de voir l'équipe d'infrastructure revoir le squelette lui-même tout en conservant les interfaces des fonctionnalités opérationnelles (ou en forçant les mises à jour). Ce genre de restructuration représente l'un des coûts résultants de l'utilisation de cette stratégie.

Le développement micro-incrémental, c'est lorsque les équipes intègrent leur travail chaque heure, demi-journée ou journée. Plus le temps est court entre les intégrations, plus le délai de détection des erreurs est court, et plus le coût pour faire des changements est réduit. L'autre avantage, c'est qu'il y a moins de risques que les membres de l'équipe modifient la même partie du code en même temps, qu'ils ont moins besoin de récupérer le code des autres membres de l'équipe du dépôt et de faire des branches, rendant ainsi l'intégration du code plus facile, rapide et moins sujette à l'erreur.

Un piton [6, 7] est une petite quantité de travail jetable créé explicitement pour répondre à la question « Allons-nous trouver un problème en utilisant cette approche ? ». Le piton est utilisé pour débusquer les problèmes d'interfaces, de performances et les problèmes de passage à grande échelle.

La différence entre un piton et un développement incrémental ordinaire est que ce dernier est conduit en utilisant l'ensemble des conventions de production, avec le postulat que le travail effectué sera utilisé dans le produit final. Un piton ne doit absolument pas être utilisé dans le produit final : c'est du travail jetable. Ce travail étant jetable, cela signifie qu'il est toujours fait de la manière la plus rapide et la plus efficace possible dans le seul but d'en apprendre le plus possible par rapport à la question du moment.

Certaines questions semblent impossibles à faire avancer dans les délais impartis, comme la conversion d'une base de données. Avec le fractionnement des *user stories* [8], une *story* peut être découpée d'une part en un fragment (piton) de connaissance à acquérir (l'apprentissage) et d'autre part en un fragment qui ira en production. Le piton est planté alors très tôt pour savoir quelles difficultés pourraient surgir. Le travail proprement dit (après apprentissage – NdT) peut-être mis de côté jusqu'au moment approprié par rapport au calendrier du projet.

Apprendre à travailler ensemble

Parfois, le fait d'échouer à faire une livraison peut être dû non pas aux gens d'avoir fait incorrectement leur travail, mais de ne pas avoir appris à travailler ensemble. Dans ce dernier cas, Tom DeMarco et Tim parlent d'une « équipe gélifiée ». Trois stratégies permettent de créer une équipe gélifiée :

- La victoire anticipée
- Le squelette ambulant
- Le plus simple d'abord, le pire ensuite

La stratégie de la « victoire anticipée » [10] se base sur les travaux du sociologue Karl Weick [11] montrant que terminer quelque chose permet aux gens de se faire confiance, de leur redonner du moral et d'être plus performants.

Le « squelette ambulant » décrit précédemment permet d'offrir une victoire technique anticipée à l'équipe et aux sponsors. Le concept est parfois légèrement modifié pour implémenter et livrer un fil très tenu qui court tout le long du flux de travail d'une entreprise, c'est en quelque sorte un mélange de « victoire anticipée » et d'apprentissage technique sur les aspects liés à la livraison et à la manière de travailler au sein du projet.

La stratégie « le plus simple d'abord, le pire ensuite » [12] est contraire à la recommandation habituelle préconisée dans le monde du développement agile. La recommandation agile habituelle est de réaliser tout d'abord la valeur métier la plus élevée. Cette stratégie a du bon sens une fois que l'équipe fonctionne bien, que les risques sociaux ont été diminués, que l'équipe est capable et a confiance en sa capacité, de livrer n'importe quel élément ayant une valeur métier élevée.

Toutefois, de nombreuses discussions devront avoir lieu avant que l'équipe atteigne ce point. Pour cette raison, il est utile de temps en temps de réaliser quelque chose de concret mais de très simple afin que les membres de l'équipe puissent ajuster leurs habitudes sociales à temps avant que les parties difficiles ne soient atteintes.

Apprendre combien cela va coûter

Deux stratégies aident à connaître le coût d'un projet :

- Les carottes d'échantillons
- Le microcosme

Tim Lister a raconté l'histoire suivante lors d'une conférence [13] : « un homme souhaite qu'un bassin soit construit dans son jardin derrière sa maison, il invite trois entrepreneurs à lui présenter des devis. Le troisième entrepreneur, à la place de présenter un devis, indique au propriétaire qu'il aurait besoin de forer et de récupérer une carotte d'échantillon du sol, et qu'il facturera le propriétaire pour cela. Le propriétaire récrimine, et dit que les deux premiers entrepreneurs ne l'ont pas fait payé pour récupérer des échantillons du sol. L'entrepreneur répond qu'il n'a aucune idée de comment les deux premiers entrepreneurs pourraient soumettre un devis, étant donné qu'ils ne connaissent pas la nature des couches rocheuses en-dessous de la pelouse, mais que lui ne pourrait pas faire de devis sans avoir cette information. Désormais en confiance avec le troisième entrepreneur, le propriétaire choisit de le retenir pour réaliser ce travail. »

Pour faire cela avec une équipe de développement, isolez donc les parties du système pour lesquelles le développement ne paraît pas évident et développez alors de très petits éléments dans ces parties-là. Au cours de ce développement, identifiez quelles sortes de surprises se cachent sous la surface et ainsi avoir une compréhension du niveau de difficulté qui sera rencontré lors du projet. Sélectionnez avec soin le type de « carottes d'échantillons » qui permettra à l'équipe de développer une estimation plus fiable en terme de coût, de délai et de ressources pour le projet.

Le carottage d'échantillon est une version miniature d'une stratégie plus générale appelée « microcosme » [14]. Cette stratégie consiste à lancer un mini projet dans le seul objectif d'établir une estimation correcte. Un projet microcosme peut être lancé pour tester la productivité d'une nouvelle équipe de développement (pensez aux équipes *off-shore* en particulier), aussi bien que pour tester la vitesse d'apprentissage des équipes existantes avec une nouvelle technologie, pour comparer la productivité des personnes expertes par rapport à des personnes non-expertes ou à des nouveaux développeurs.

Alors qu'un carottage d'échantillon est prévu pour prendre quelques heures ou quelques jours, un projet microcosme peut prendre des semaines à mener à terme, et ne devrait être entrepris que dans le cadre de développements de grande envergure.

Établir un planning

À la lumière de ces stratégies, la création d'un planning projet se fait de manière très différente par rapport à ce qui se faisait auparavant.

L'apprentissage discipliné nécessite de fusionner les étapes d'apprentissage issues des quatre catégories évoquées ci-dessus avec les exigences de croissance de la valeur métier comme cela est la norme avec le développement incrémental. La valeur métier et l'apprentissage sont savamment entrelacées en une séquence d'attribution de tâches conçue pour réduire le risque, livrer des informations cruciales, et développer les capacités du produit d'une manière « optimale ».

C'est là où la créativité entre en jeu.

La qualité du planning est fonction de la capacité des planificateurs à identifier, à fusionner les besoins d'apprentissage et les possibilités à venir pour en tirer des gains. Au fur et à mesure que des leçons en sont tirées, que de nouveaux risques et opportunités sont remarquées, le projet devra être mis à jour.

Ajuster la queue

La fonctionnalité d'un produit est composée en fait de trois parties :

- L'apprentissage
- La valeur
- La queue

La « queue » regroupe le polissage et le lustrage qui rend une fonctionnalité « fantastique ». Étant donné que toutes les fonctionnalités ne sont pas d'égales valeur aux yeux des acheteurs et des utilisateurs, certaines ou même la plupart des fonctionnalités peuvent être réduites ou ajustées sans endommager le système.

En présence d'une queue, une équipe peut s'arranger pour qu'un ensemble minimum de fonctionnalités soit à un niveau « adéquate » de « fantastique » longtemps avant la livraison finale, puis de passer le reste du temps à polir et à lustrer les fonctionnalités les plus importantes [15]. De manière alternative, si le temps est réduit, l'équipe peut faire l'impasse (ajuster) sur le polissage et livrer de manière anticipée ou dans les délais impartis [16]. La dernière section du présent article donne davantage d'informations à ce sujet.

En récolter les bénéfices

L'apprentissage discipliné présente deux bénéfices : des gains anticipés et la capacité d'ajuster la queue. Les gains anticipés issus du développement incrémental sont très bien décrits dans *Software by Numbers* [17]. Un projet peut devenir auto-financé en livrant des éléments aux acheteurs pendant son développement, permettant ainsi de diminuer la charge du financement pour les sponsors.

Moins évident mais tout aussi valable est la capacité de ne pas développer les éléments du système ayant le moins de valeur. Voici un bref exemple pour vous en donner une idée :

Lorsque vous ouvrez un nouvel hôtel, il n'est pas vraiment nécessaire de faire briller les poignées de portes avant d'ouvrir au public. S'il est nécessaire d'avoir des poignées de portes qui brillent pour les clients, il n'est probablement pas nécessaire que toutes les poignées de portes aient à briller.

Vous pouvez ajuster n'importe lequel des quatre aspects d'un système :

- Les fonctionnalités
- Le détail des fonctionnalités
- La qualité d'utilisation
- La qualité interne

Vous pouvez laisser tomber toute une fonctionnalité. Une voiture (par exemple) peut ne pas avoir besoin d'un toit ouvrant. Les premiers iPads n'avaient pas de modem.

S'il ne s'agit pas d'une fonctionnalité dans son entier, vous pourriez ajuster tel ou tel aspect d'une fonctionnalité : étant donné que votre voiture doit avoir tous les éléments de base (tels que les freins), elle n'a pas peut être pas besoin d'avoir un système d'anti-blocage. Un système informatique pourrait avoir besoin d'une fonctionnalité de recherche, mais pas d'auto-complétion ou d'auto-correction.

Prendre en compte cela, autrement dit rendre l'utilisation des fonctionnalités plus facile et plus fluide demande énormément de travail, vous pourriez donc être amener à choisir de faire l'impasse sur l'amélioration de l'utilisabilité sur certaines fonctionnalités.

Enfin, vous pouvez ajuster la qualité et la pertinence de la conception interne des fonctionnalités. Autrement dit, la question est quel est le niveau de qualité interne nécessaire pour cette livraison en particulier.

Si le développement a été fait de manière incrémentale, et en fonction des domaines d'apprentissage, l'équipe alors peut livrer :

- De manière anticipée, des fonctionnalités limitées ou un niveau de qualité réduit
- Dans les délais impartis, avec un niveau de qualité optimal ou réduit selon le degré d'avancement du projet à ce moment-là
- Ou plus tard, avec des fonctionnalité enrichies ou un niveau de qualité supérieur selon ce qui aura été choisit par les sponsors !

Dans les circonstances habituelles d'un projet, les seuls choix possibles sont livrer en retard ou faire des heures supplémentaires. L'option « ajuster la queue » n'est valable que pour les équipes sachant travailler de manière disciplinée.

L'apprentissage discipliné conjugué avec l'ajustement de la queue est l'une des quelques approches pouvant être faites sur de très petits comme sur de très gros projets, des projets à prix fixe (forfait) ou à prix variable. Voici trois exemples tirés de cas réels :

1. Petit projet à prix variable : le développement d'un site web impliquant uniquement le propriétaire du site web et le développeur. Après quelques mois de travail, le propriétaire du site web souhaite que le site soit livré « bientôt », il ajuste donc la queue de manière agressive et répétée jusqu'à arriver à quelque chose de beaucoup plus petit que prévu mais qui s'avère tout à fait suffisant pour un déploiement.

2. Petit projet à prix fixe (forfait) : l'entreprise dont il est question ici a pour habitude de faire des propositions commerciales à très bas coûts, avec des contrats à prix fixe de trois à six mois, impliquant trois à huit personnes. Ces propositions sont faites toujours dans une approche commerciale agressive avec des équipes finissant presque toujours en retard, ratant la date butoir ou ne respectant pas le périmètre, ce qui a pour conséquences des heures supplémentaires pour les développeurs et des pénalités de retard pour l'entreprise à la fin du contrat. Jeff Patton [18] a travaillé comme le décrit le présent article, en mettant les fonctionnalités les moins importantes à la fin et en réduisant délibérément les fonctionnalités les moins critiques, de telle manière qu'à la fin du contrat, les clients sachent qu'ils ont obtenu la plus grande partie de ce qu'ils voulaient. C'est en travaillant de cette manière qu'il y a le moins d'heures supplémentaires, le moins de pénalités, la plus grande satisfaction client et la plus grande probabilité d'avoir un nouveau contrat pour la suite.

3. Très grand projet de développement : voici une autre entreprise avec plusieurs milliers de développeurs répartis à travers le monde, travaillant sur une ligne produits ayant plusieurs variantes, applications et versions. Dans des circonstances normales, lorsqu'il est nécessaire de faire une intégration de l'ensemble à une date donnée, chaque équipe commence à faire des heures supplémentaires et se bat pour ne pas être l'équipe qui sera en retard sur le calendrier. La date d'intégration glisse au fur et à mesure que les équipes les unes après les autres échouent à réaliser leurs travaux dans les temps. En utilisant l'approche d'ajuster la queue chaque équipe aurait pu mettre en place les éléments essentiels nécessaire pour l'intégration, avec uniquement en queue les éléments non terminés. Pour la livraison, le management serait donc en mesure de décider de livrer un peu moins, dans les temps, ou un tout petit peu plus tard.

C'est vraiment très intéressant de trouver une stratégie de référence qui peut s'appliquer à des projets ayant des tailles et des natures vraiment très différentes comme cela a pu être souligné.

L'apprentissage discipliné, ce n'est pas pour les mauviettes. Il exige de la discipline, de la créativité et des ajustements constants. Les bénéfices pour une équipe, c'est la capacité à travailler ensemble, de découvrir ce qui est nécessaire dans les temps, de livrer de manière anticipée afin d'arriver à avoir un projet qui s'auto-finance et enfin ajuster la queue pour arriver à répondre à des dates butoirs fermes.

À PROPOS DE L'AUTEUR

Le Dr. Alistair Cockburn est l'un des créateurs du manifeste pour le développement agile de logiciels et a été élu en 2007 comme l'un des "150 héros de tout les temps des technologies de l'information" pour son travail avant-gardiste sur les cas d'utilisations et le développement agile de logiciels. Il est un stratège renommé sur les technologies de l'information et l'auteur des livres "*Agile Software Development*" et "Écrire des cas d'utilisations efficaces" ayant tous les deux remportés le prix Jolt. Il est un expert sur le développement agile, les cas d'utilisations, le processus de conception. En 2001, il est le co-auteur du manifeste agile, en 2003, il a créé la *Agile Development Conference*, en 2005 il a co-fondé le *Agile Project Leadership Network*, en 2010 il a co-fondé le *International Consortium for Agile*. Nombre de ses articles, interventions, poèmes et blog sont accessibles sur <http://alistair.cockburn.us>.

E-mail: totheralistair@aol.com

RÉFÉRENCES

1. <https://fr.wikipedia.org/wiki/Maquettage_papier>
2. <https://en.wikipedia.org/wiki/Use-centered_design>
3. BBC "Searching the internet's long tail and finding parrot cages," <http:/ /www.bbc.co.uk/news/business-11495839>
4. Reis, E., The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011.
5. <http://alistair.cockburn.us/Walking+skeleton>
6. <http://c2.com/xp/SpikeSolution.html>
7. <http://agiledictionary.com/209/spike/>
8. <http://alistair.cockburn.us/The+A-B+work+split>
9. Tom DeMarco and Timothy Lister, Peopleware: Productive Projects and Teams, New York: Dorset House Publishing Co., 1987.
10. <http://alistair.cockburn.us/Advancedpmstrategies1-180.ppt>
11. Karl Weick, The Social Psychology of Organizing, McGraw-Hill Humanities/ Social Sciences/Languages; 2nd edition, 1979.
12. Alistair Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley, 2005. Also online at <http://alistair.cockburn.us/ASD +book+extract%3A+%22Individuals%22>
13. Lister, Tim, keynote at Agile Development Conference 2010.
14. <http://alistair.cockburn.us/Project+risk+reduction+patterns>
15. <http://www.agileproductdesign.com/downloads/patton_embrace_ uncertainty_optimized.ppt>
16. <http://alistair.cockburn.us/Trim+the+Tail>
17. Mark Denne and Jane Cleland-Huang. Software by Numbers: Low-Risk, High-Return Development. Prentice-Hall, 2003.
18. Jeff Patton, "Unfixing the Fixed Scope Project: Using Agile Methodologies to Create Flexibility in Project Scope," in Agile Development Conference 2003, Proceedings of the Conference on Agile Development, 2003, ACM Press. Available online through a Google docs search.