
L'eXtreme Programming

Pour tenter de comprendre en quoi cette méthode est extrême pour les néophytes mais indispensable pour les initiés...



:: AVERTISSEMENT ::

Ce document, qui présente l'*Extreme Programming*,
est rédigé en respectant l'un de ses principes :
Itératif et Incrémental.

Ainsi, cette présentation actuellement dans un état inachevé,
est complétée chaque semaine (l'itération)
d'un nouveau chapitre (l'incrément)
afin de recueillir fréquemment le feedback du client (vous).

Vous pouvez obtenir la dernière version à l'adresse suivante:

<http://social.hortis.ch/wp-content/uploads/2006/11/extreme-programming.pdf>

v06-a - 23 novembre 2006

20 pages

Table des matières

Table des matières

Table des matières.....	2
Présentation	3
Les « 4 » valeurs.....	5
La communication.....	5
Le retour d'information ou feed back.....	6
La simplicité.....	7
Le courage.....	8
Le respect.....	8
Conclusion.....	8
Les « 13 » pratiques.....	10
Le client sur site.....	11
Les livraisons fréquentes.....	13
La métaphore.....	17
Le Planning Game.....	19
Contrat Creative Commons.....	20

Présentation

XP a été mise au point par "trois mousquetaires" (et pas des moindres), **Kent Beck**, **Ward Cunningham** et **Ron Jeffries**, durant un projet de facturation chez Chrysler, le « *Chrysler Comprehensive Compensation System* » ou C3. Kent Beck en devient le chef de projet en 1996, redéfinit la méthodologie du projet en raison des mauvais résultats, commence à la formaliser et en Octobre 1999 « *Extreme Programming Explained* » est publié.

Ainsi, XP qui, en tant que méthode agile, **prône l'empirisme et l'apprentissage, est donc née de l'Expérience.**

Dés sa naissance, **XP s'est voulue simple, dépouillée.** Il s'agit d'un regroupement de "**seulement**" **13 pratiques** qui, comme la plupart du temps pour les méthodes Agiles, **tiennent du bon sens** : XP n'a rien inventé, elle ne fait qu'associer un ensemble de pratiques "vitales" et complémentaires mais préexistantes.

On peut donc résumer XP et ses pratiques sur un *post-it*. Cela permet souvent à ses détracteurs d'argumenter que XP est une méthodologie simpliste, qui ne permet pas de couvrir l'ensemble des problématiques que l'on peut rencontrer dans le développement logiciel et, à fortiori, est incapable d'amener un projet à terme en tenant les délais et les objectifs.

Dans la philosophie XP, il est plus aisé d'appliquer et maîtriser un nombre restreint de règles plutôt que d'avoir à choisir parmi une longue liste se voulant exhaustive... De plus, les probabilités de dérives augmente avec le nombre de règles à appliquer, voire à adapter.

XP est une méthodologie qui s'intègre dans un mouvement plus global dit **Agile**. L'objectif des **méthodes agiles**, Scrum ou XP pour les plus connues, est de **se focaliser sur les besoins du client**, et de **considérer le changement comme une composante** entière du développement logiciel. Mais pour que cela fonctionne, les deux parties doivent s'investir dans un **contrat gagnant-gagnant...**

Pour assimiler sans douleur le changement XP, comme toute méthode agile, valorise la *feedback via* un processus itératif et incrémental.

Pour bien différencier ces deux notions clefs, je vous renvoie à l'excellent article de Stéphane : « [Itératif ET Incrémental](#) »¹.

Le projet, **itératif**, est donc une succession d'itérations de durée constante. On parle aussi de boîte de temps. Mais cette durée doit être appropriée au projet (taille, complexité, domaine métier...) : on choisira une longueur de 1 à 4 semaines, 2 semaines étant souvent un bon compromis (par expérience). Le but est en effet d'être

1 <http://social.hortis.ch/2006/08/17/iteratif-et-incremental/>

capable de livrer un « bout » de logiciel, pertinent et fonctionnel, en une itération.

En plus d'être itératif, XP est **incrémental**. Cela signifie qu'à la fin de chaque nouvelle itération, le morceau d'application livré est pourvu de quelques fonctionnalités supplémentaires.

Pour identifier ces nouvelles fonctionnalités que l'on va ajouter dans l'**itération** à venir, chaque itération commence par un **planning game** (d'environ un jour).

Le **client** y présente les **scénarios** à réaliser, puis **l'équipe** découpe chaque scénario en **tâches** et enfin évalue la durée de chacune de ces tâches. Le client peut alors « prioriser » ses scénarios (définir les priorités selon l'importance qu'il leur donne). Il pourra alors identifier ceux qui seront délaissés, en fonction de ces évolutions et du nombre de jours-binôme disponibles pour l'itération.

Puis les développements peuvent reprendre.

Quand on réalise une fonctionnalité, on commence par écrire le **test release** associé. Ensuite, on peut réaliser les tâches en commençant par développer... les **tests unitaires**. Pour cela, chaque matin, au **stand-up meeting** (d'environ 15 minutes), on définit les **binômes** pour la journée et cherche à identifier les éventuels problèmes qui pourraient empêcher la livraison en fin d'itération.

Tels sont les événements qui rythment la vie d'un projet XP...

Voyons cela plus en détail. Afin d'assurer au quotidien un bon fonctionnement, et le cas échéant de garantir la remonté le plus tôt possible de tout dysfonctionnement, XP vient avec des valeurs, qui se déclinent concrètement en 13 pratiques dont nous venons d'en apercevoir quelques unes. XP définit également quelques rôles clefs afin de répartir les rares fonctions qui se sont pas (encore) partagées et maîtrisées par l'équipe... Cette présentation repose sur la première version, épurée, d'XP. Elle serait donc incomplète si je n'abordais pas "La Seconde version d'XP"...

Les « 4 » valeurs

Pour fonctionner, XP nécessite 4 éléments fondamentaux identifiés par Kent Beck, qui doivent absolument se retrouver dans chaque membre de l'équipe pour un développement logiciel réussi.

La communication

La plupart des problèmes qui surviennent sur un projet sont issus d'un manque de communication.

Cette « vérité est tellement vraie » que cette valeur sort du cadre XP et se retrouve dans bien des formations à la gestion de projet en particulier, et au management en général.

La communication doit donc être valorisée, voire maximisée, aussi bien horizontalement que verticalement : tant entre les membres de l'équipe, qu'entre l'équipe et le client.

Pour cela, chacune de ces deux dimensions est déclinée en pratiques :

- « Client sur site » et « Livraisons fréquentes » pour la communication avec le client
- « Programmation en binôme » pour la communication entre membre de l'équipe
- « Planning game » et « Stand up meeting » pour les deux cas.

Mais que veut dire « optimiser la communication entre des personnes » ? En quoi consiste cet effort incessant ?...

La Connaissance, explicite ou tacite, doit se transmettre oralement : elle doit passer directement de la tête au code.

Par écrit, la connaissance est plus facilement altérée, désynchronisée.

Par oral, associée au courage, cette connaissance sera précisée, affinée voire complétée. Dans tous les cas elle sera instantanée.

On pourrait même rajouter une troisième dimension à la communication : entre les membres de l'équipe et les artefacts du projet, à savoir tout ce qui est produit par l'équipe.

Dans ce cas, une bonne communication est synonyme d'artefacts « facilement lisibles

» et à jour !

Là encore, cela se décline en pratiques : l'« Intégration continue », les « Tests (recette et unitaires) », la « Conception simple », la « Métaphore ».

Un bon moyen de lutter contre des artéfacts non à jour, périmés, est de favoriser la capture de connaissance métier en code source, et non en document dans son outil de bureautique favori : un document texte aura plutôt tendance à croître avec le temps, et avec lui les portions de texte non relues/validées.

Le retour d'information ou feed back

L'objectif premier des méthodes agiles, est de se focaliser sur le besoin du client : le logiciel.

Il est donc important dans cet état d'esprit de pouvoir interroger celui qui est le mieux renseigné sur le véritable état du logiciel : à savoir le logiciel lui-même.

Cela afin de pouvoir mesurer à quelle distance on est de l'état final, et identifier au plus tôt toute digression.

Il faut valider ce qui a été fait, rapidement et donc fréquemment. Cela permet de faire évoluer l'existant avec confiance et assurance, ou dans le cas contraire d'identifier une erreur au plus tôt, afin de remonter facilement à son origine, et donc faciliter sa correction.

Les feedbacks importants sont ceux attachés aux trois dimensions de la Communication identifiées ci-dessus : entre membres de l'équipe, avec le client et avec les artéfacts.

En fait, feedback et communication sont intimement liés. Il faut communiquer afin d'obtenir un feedback. Les résultats de ce feedback peuvent ensuite alimenter d'autres communications...

Basée sur ce principe, on retrouve dans les projets [[XP]] une autre notion : les « "Big visible charts" ». L'idée est d'afficher de grands graphiques, régulièrement mise à jour, basés sur une des métriques, rapidement compréhensibles. Le but est de mettre en relief d'éventuel problème et, le cas échéant, de favoriser l'implication, la compétition saine.

Le feedback permet également de faire émerger et de maintenir de la Simplicité.

Un bon moyen de générer et de conserver de la Simplicité est d'avoir une approche empirique : j'essaye, puis je corrige mes erreurs.

Or, pour que cela fonctionne il faut un feedback fréquent, pour ne pas dire permanent. Et, plus le système est simple, plus il est facile d'obtenir du feedback : nous voilà enfermer dans un « cercle judicieux »...

La simplicité

« *Do the simplest thing that could possibly work* ».

La conception, l'architecture et les développements doivent être simple, et constamment simplifiés.

Antoine de Saint-Exupéry semblait partager cette approche : « *La perfection est atteinte non quand il ne reste rien à ajouter, mais quand il ne reste rien à enlever* ».

Cela facilite l'assimilation pour tout nouveau développeur, et la réalisation des évolutions et modifications à venir.

En réduisant le code, la Simplicité en accroît la qualité et favorise la Communication.

Une bonne ligne de conduite est de « ne pas tirer de plans sur la comète » : on ne développe pas de fonctionnalité non demandée, non nécessaire, ou facilitant la réutilisation, aussi géniale soit-elle !

Si le système est simple, l'ajout ou la réutilisation d'une fonctionnalité quand nécessaire se fera facilement.

Mais il ne faut pas confondre « simple » et « simpliste » !

Cette valeur est assurément l'une des plus difficile à définir et quantifier : elle demande de l'expérience, de l'imagination, du travail et de la discipline. Donc du Courage...

Le courage

Au quotidien, le développeur XP est confronté à toute sorte de difficultés : techniques, méthodologiques, relationnelles...

Il doit constamment être ouvert, se remettre en question, s'exposer, prévenir quand il ne sait pas, si il a fait une erreur, chercher à comprendre... Pour surmonter ces situations inconfortables, les dépasser et améliorer, le développeur doit s'armer de courage.

La motivation même des Méthodologies, des Processus et autres Protocoles est de maîtriser et réduire le Risque, donc les craintes.

Plus nos craintes sont grandes, sur un projet logiciel notamment, plus ces Méthodologies doivent être robustes.

Cela est d'autant plus vrai avec les Méthodes Agiles, qu'elles intègrent le changement comme une composante naturelle !

Plus nous favorisons la Communication, le Feedback et la Simplicité, moins de courage nous avons besoins, même et surtout en cas de changements dans les besoins ou de remaniement dans le système.

Le courage est d'autant plus important en XP, que bien souvent, les fois où l'on donne sa chance à la controversée XP sont celles où le projet va mal et que l'on a plus rien à perdre. Les conditions sont alors loin d'être optimales.

Le respect

Le respect étant apparu dans la seconde édition de « XP explained », j'aborderai cette valeur dans le chapitre consacré à la 2e version...

Conclusion

Comme je le dis en introduction, ces valeurs doivent se retrouver dans chacun des membres de l'équipe...

Ainsi, en cas d'embauche pour former ou compléter une équipe XP, demander au

candidat quelle est sa vision pour chacune de ces valeurs, peut être un bon moyen de l'évaluer...

Aller, je joue le jeu : je vais faire preuve de Courage en essayant de donner mon Feedback le plus Simplement possible sur les valeurs XP.

Pour moi, si je devais décrire les valeurs XP :

- *feedback = a x communication*
- *feedback = b x simplicité*
- *feedback x communication x simplicité = c / courage*

Aucune ne peut donc être nulle, surtout pas le courage !

- *respect = t ^ d*

Nulle au départ, cette valeur prend rapidement de l'importance avec le temps

Ces valeurs, qui permettent d'acquérir un état d'esprit positif, ne donnent pas de conseils précis quand à la gestion du projet ou le développement du logiciel.

C'est pour cette raison qu'elles sont déclinées en 13 pratiques...

Les « 13 » pratiques

XP décline donc ces valeurs et principes en 13 pratiques qui vont du développement au processus.

De façon plus précise, ces pratiques se regroupent en 3 catégories :

- Programmation
- Collaboration
 - La métaphore
- Gestion de projet :
 - Les livraisons fréquentes
 - Le client sur site

A titre de comparaison avec d'autres méthodes Agiles, il faut savoir que XP « impose » ces pratiques alors que Scrum par exemple les considère comme de bonnes pratiques à mettre en oeuvre selon le besoin.

Pour chacune de ces pratiques, en plus de la présentation, je vais essayer, dans la mesure du possible, de retranscrire les discussions, les différentes interprétations ou mise en oeuvre, voire les polémiques, que se soit entre praticiens ou détracteurs d'XP. En effet, je trouve cette approche aussi originale qu'intéressante. Le fait de résumer ces variations, rencontrées sur les mailing-list, les sites, les événements ou rencontres, permettent de se faire une meilleure idée de la pratique abordée et donc de mieux la mettre en oeuvre. J'ai nommé ces chapitres « Paul et Mick dans l'extrême »... Désolé !

Le client sur site

Idéalement, le client XP est une personne de l'équipe cliente. Concrètement, il peut être un développeur qui abandonne le développement pour embrasser le rôle de client XP.

Durant le "planning game" le client a comme fonction d'exposer les scénarios qu'il souhaite voir réalisés durant l'itération. En fin de "planning game", le client doit leur attribuer des priorités, une fois ces scénarios estimés par les développeurs, afin d'identifier ceux qui tiennent dans l'itération.

Durant l'itération, le fait qu'il soit sur site (dans le même espace que l'équipe, dans ce que l'on appelle la « *war room* »), lui permet d'être directement accessible par les développeurs pour des questions, et donc les réponses. Cela implique que le client XP ait à la fois la connaissance pour répondre, mais aussi le pouvoir de prendre des décisions (ne pas faire telle fonctionnalité, la faire autrement, ou après telle autre...).

Idéalement le client écrit les tests release avec un binôme de développeurs. Cela impliquant un formalise de capture de ces tests qui soit de haut niveau, en pratique le client se contente souvent de les concevoir, et se sont les développeurs qui les écrivent.

Il faut bien comprendre que cette pratique a pour but d'améliorer le *feedback*, de raccourcir le délais entre une remarque du client et la prise en compte par l'équipe (dans un sens) ou une question de l'équipe et la réponse du client (dans l'autre sens).

XP n'est pas opposée à la rédaction de documents. Elle la considère simplement comme un moyen d'atteindre un but : la capture et le transfert de la connaissance. Mais XP lui préfère d'autres moyens : le client sur site en est un. Maintenant si le client a un réel besoin de documentation, on pourra toujours identifier une tâche à cet effet, la documentation devenant alors, et dans ce cas seulement, une partie intégrante de ce qu'il faut livrer au client, de son besoin réel.

Il faut également avoir à l'esprit que le travail de réflexion et d'appropriation du métier qui se fait d'habitude pendant la réalisation du document de spécifications fonctionnelles, mais aussi celui d'élaboration de l'architecture au cours de la rédaction des spécifications techniques, est dans le cas d'une équipe XP un travail de tous les instants. Il n'y a rien de figer et d'irrévocable, tant sur le plan fonctionnel que sur le plan technique. Il est donc difficilement envisageable d'avoir un client sur site qu'une

partie du temps, voire distant ou indisponible. Une question pouvant émerger à tout instant, il est optimal que le client soit constamment disponible sur site.

Paul et Mick dans l'eXtrême

Par contre, le client n'étant pas sollicité en continue, 100% du temps, mais étant une ressource comme les autres, il serait un luxe de l'affecter 100% de son temps au rôle de client. Par expérience, une configuration qui fonctionne bien est de donner la casquette de client au chef de projet. Il a bien souvent la connaissance métier. Il ne reste plus qu'à lui obtenir le pouvoir de décision.

Il pourra ainsi consacrer et donc imputer, à juste titre, un pourcentage, seulement, de son temps au rôle de client, mais être disponible à tout instant. Ce pourcentage pourra être prédéfini au début du projet, et éventuellement réajuster au cours du projet si besoin.

Toujours par expérience, il est à mon avis souhaitable que le client n'ait pas de compétences techniques, cela afin de garantir un cloisonnement des compétences, une séparation des responsabilités. J'insiste sur le fait que cela est un avis personnel que je justifie par les bénéfices de cette situation, à savoir:

- Le client n'est pas tenté de s'investir dans la conception ou le développement, voire de l'orienter ou la diriger. Chacun son métier.
- Le développeur doit, pour se faire comprendre, vulgariser ses dire, ce qui implique d'abord de les maîtriser mais aussi de les remettre en question en les abordant sous un autre angle : nous sommes très proche de la pratique de la Métaphore.
- Enfin, cela renforce le fait que les tests release doivent aborder l'application comme une boîte noire.

Toujours dans l'idée de favoriser la communication, le client participe au stand-up meeting quotidien.

J'ai parfois entendu dire qu'une trace écrite via un document de spécifications fonctionnelles validé ou un mail avec les bonnes personnes en copie, sont impératifs pour se prémunir d'un « mauvais coup » futur. Cela est un « *bad smell* » qui traduit une déviance dans la notion de contrat gagnant-gagnant. Cette problématique, dépasse le cadre de cette pratique. En effet, les deux parties peuvent limiter les dommages collatéraux, de manière contractuelle, et établir ainsi un meilleur état d'esprit, de confiance: c'est l'« *Optional Scope Contract* ».

Les livraisons fréquentes

Livrant en fin de chaque itération, le rythme des livraisons en XP est d'une toutes les quelques semaines. Il est important de s'accorder avec le client sur une fréquence donnée et de s'y tenir. Par exemple, une période de 2 semaines, pour la durée d'une itération, donc pour la durée séparant deux livraisons, est un bon compromis.

Si le client a besoin de livraisons plus fréquentes, **on peut éventuellement raccourcir cette période, mais ne surtout pas faire de livraison en cours d'itération**. Par expérience, ce « *bad smell* » traduit un dysfonctionnement dans le processus (voir le chapitre « Paul et Mick dans l'eXtrême »).

Techniquement, les livraisons fréquentes impliquent d'avoir un mécanisme d'intégration fiable, à savoir performant et automatisé.

Performant pour être compatible avec une fréquence de livraison élevé : notamment les tests release doivent se jouer relativement rapidement.

Fiable pour être capable de reproduire une livraison à partir des sources versionnées : en effet, seules les sources sont versionnées, les livraisons sont reproduites si besoin. Au-delà des sources, c'est l'environnement de développement qui doit être versionné : si l'on souhaite reproduire la livraison produite il y a 14 itérations, il faut repartir des sources de l'époque, mais aussi rétablir l'environnement de développement si l'on veut reproduire la livraison de l'époque à l'identique, avec les mécanismes de l'époque.

Un processus d'« **Intégration Continue** » efficace va de paire avec des livraisons fréquentes réussies.

La livraison est le livrable qui ponctue l'itération. Il est important de conserver cette notion de « Boite de temps ». Quoi qu'il arrive, **il est important de systématiquement produire un livrable qui fonctionne en fin de chaque itération** :

- Tant en début de projet pour valider le processus de livraison lui-même (construction, test, déploiement)
- Qu'après pour valider les nouvelles fonctionnalités.

Pour être fonctionnel, l'ensemble des fonctionnalités peut être :

- Toutes celles prévues au début d'itérations plus d'autres ajoutées après,
- Toutes celles prévues en début d'itération,
- Seulement les prioritaires selon le client lorsque les développements ont pris plus de temps que prévu.

Dans tous les cas, le client devra comprendre que, si il décide d'**ajouter en cours**

d'itération des fonctionnalités non prévues sur l'itération en cours, et il peut le faire, cela devra ce faire en deux étapes :

1. Tout d'abord l'équipe devra estimer le temps de développement de cette (ces) nouvelle(s) fonctionnalité(s).
2. Le client devra ensuite retirer des tâches de celles restant à faire (pour la fin d'itération) pour une même durée de développement : il s'agit bien d'un remplacement de tâches et non d'un ajout.

Cette pratique doit profiter de sa grande soeur, à savoir le « **Client sur site** », sans pour autant tomber dans le piège : il faut livrer, installer, déployer le logiciel sur une machine dédiée, vierge de toute installation et configuration, par opposition à une machine de développement qui est déjà configurée.

La motivation de cette pratique est de **donner au client, le plus fréquemment possible, un aperçu de l'état de l'application**, afin qu'il puisse faire un retour qui conditionnera les développements à venir : on est bien là dans une démarche itérative.

Un des effets de bord est l'installation progressive d'un climat de confiance. Quel dommage ;o)

Paul et Mick dans l'eXtrême

XP insiste donc sur le fait de fournir au client un livrable en fin de chaque itération. A titre de comparaison Scrum est moins extrême. En effet, selon la taille du projet, **une livraison ne sera intéressante que toutes les 2 ou 3 itérations**. On pourra alors fournir au client le livrable au bout de 2 ou 3 itérations. Mais on continuera de **produire à chaque fin d'itération un livrable qui vérifie les tests release** identifiés en début d'itération, afin de **rester dans une démarche incrémentale**.

L'équipe doit vraiment voir cette pratique comme une chance : celle d'avoir le plus tôt possible **la possibilité de corriger d'éventuelles erreurs ou mauvaises interprétations**.

Considérons le projet comme la période d'essai d'un nouvel embauché. Le but est d'avoir un avis positif à la fin de la période d'essai, que l'employeur soit satisfait afin qu'il fasse de nouveau confiance pour la suite... Et bien les livraisons fréquentes sont autant de « bilans intermédiaires » qui visent à redresser le tir et augmentent les chances de satisfaction finale.

Mais il ne faut pas confondre fréquence et précipitation. J'ai rencontré un client qui nous a imposé durant une période de crise de livrer tous les 3 ou 4 jours (sur des itérations de 3 semaines), nous demandant de court-circuiter la phase d'intégration et de tests afin de tenir cette fréquence. Il nous a alors reproché les bugs inhérents aux livraisons, la baisse de qualité des livrables étant indéniable. Il nous imposa alors de corriger les nouveaux bugs tout en tenant la nouvelle fréquence : nous nous sommes

alors enfermé dans cette logique de « **livrer plus souvent principalement pour corriger les bugs qui apparaissent** ».

La difficulté dans cette situation est d'arriver à faire comprendre au client qu'il perdra plus de temps, et donc plus d'argent, à corriger les bugs générés, qu'il pensait en gagner.

De plus, cette perte d'argent s'accompagne d'une perte de qualité dans l'image de marque, donc de confiance, ce qui est tout aussi déplorable.

La correction se fait au prix des autres fonctionnalités : d'où l'importance d'estimer toute demande de changement, notamment la correction de bug, afin de quantifier auprès du client les fonctionnalités qui ne seront délaissées en contrepartie.

Cette dynamique de livrer pour corriger des bugs est présente sur des projets non XP, où il n'y pas la pratique des « Livraisons fréquentes ».

L'autre extrême est tout aussi préjudiciable. Pas plus tard qu'hier, j'ai entendu dire : « **ça n'a pas de sens de faire une livraison intermédiaire sur ce projet** ». Plus qu'un « *bad smell* », cela traduit à mon sens un (mauvais) conditionnement de l'esprit. Il est évident qu'une maison constituée de juste 4 murs avec les ouvertures pour les portes et les fenêtres n'a pas de « sens fonctionnel » : elle n'est pas habitable. Mais le but de cette « livraison intermédiaire » n'est pas d'y habiter mais de détecter d'éventuelles erreurs grossières au plus tôt, telles qu'une mauvaise orientation de la façade, ou une mésentente dans les dimensions des ouvertures ou la surface habitable. Parmi les quelques personnes que je connais et qui ont fait construire une maison, TOUTES SANS EXCEPTION, allaient constater l'état d'avancement du projet au moins une fois par semaine : ce n'était pas pour y habiter !

Il ne faut pas confondre le but d'une livraison intermédiaire (s'assurer que tout le monde travail dans la bonne direction) et celui de la livraison finale (satisfaire le besoin du client).

Enfin, la fréquence permet de diminuer la quantité de travail à valider.

Je vais encore prendre un exemple. J'ai récemment dû rédiger un document de spécifications fonctionnelles sous forme de « *Use Cases* ». Si je vous livre chaque semaine, une dizaine de pages correspondant à 2 ou 3 *Use Cases*, j'obtiendrais aisément de vous une relecture hebdomadaire, régulière, qui deviendra plus aisée à mesure que vous appréhendez la formule, que vous assimilez l'exercice. De plus, vous pourrez mûrir votre compréhension, et donc avoir un jugement de plus en plus critique. Maintenant si je débarque au bout de 2 mois avec un document de 80 pages se voulant complet, dont vous découvrez le fond et la forme pour la première fois... Ai-je une chance d'avoir un retour ? Voire une validation ? Et si oui, en temps et en heure ? Quelles en seront alors la finesse et la pertinence ? Il ne faut pas négliger cet aspect de l'effort nécessaire, que j'ai bien envie d'appeler le paradigme de « l'escalier et de la corde raide ». La tour « Taipei 101 » est actuellement la plus haute tour du monde avec ses 508 mètres : il est envisageable de la gravir via son escalier, mais

pensez-vous envisageable de la franchir d'un trait avec une corde raide ?

Au final, cette pratique des « **Livraisons fréquentes** » s'avère bien plus consensuelle, bien moins extrême que ce que l'on peut rencontrer traditionnellement sur les projets informatiques.

IN PROGRESS



La métaphore

La motivation de cette cette pratique XP, est de parvenir à **décrire un module de l'application, voire l'application dans son ensemble, en utilisant une analogie**, le champ lexical d'un autre domaine : métier, loisir, vie courante...

Trouver une métaphore pertinente **prend du temps**.

Aussi, on pourra poser une ou deux réunions d'une demie heure par exemple, pour identifier une métaphore qui s'adapte judicieusement à l'application. On pourra éventuellement profiter du **Planning Game** pour faire cet exercice. Dans ce cas, il faut profiter du fait que tous le monde soit réunis sans pour autant en abuser et déborder sur « l'ordre du jour » du "planning game" : une demie heure au cours de deux ou trois *planning game*. Ensuite, on pourra utiliser et affiner cette métaphore au quotidien dans les développements, la communication avec le client, avec un nouveau membre de l'équipe...

Quels sont les intérêts ?

Tout d'abord pour l'équipe de développement, l'exercice en lui-même d'identifier une métaphore oblige à **mettre à plat la connaissance sur l'application**, donc le domaine métier comme la conception. Ainsi on **diffuse et homogénéise** la connaissance, on lève les ambiguïtés, comble des lacunes voire éventuellement assaini une portion insatisfaisante de la conception lors du « **Remaniement Continu** ».

La mise en place d'une métaphore est donc un prétexte à communiquer.

Le développeur étant souvent un grand enfant joueur, il se prête volontiers aux petits jeux intellectuels, l'association d'idées en étant un parmi d'autres... Une classe de log qui reçoit des informations brutes et les affiche de façon compréhensible, c'est un peu comme une télévision... Si on appelait cette classe `Television`. Mais si l'antenne est débranchée ou que le câble est coupé, il faut un moyen de faire la différence entre cette panne et une absence de programme télé afin que l'utilisateur agisse. Peut être qu'une classe `Maintenance` ou `Repairer` Ca y est, on file la métaphore... Et innocemment, **on réfléchi en s'amusant...**

Une fois établie, la Métaphore devient un média de communication, sur les fonctionnalités de l'application et/ou sa conception. De plus, cette métaphore pourra très bien se révéler dans l'implémentation même, dans le code, dans le nom d'un

module, d'une classe, d'une fonction, d'une structure...

Enfin, l'idéal est de choisir un champ lexical facilement accessible, compréhensible pour tout nouvel arrivant, qui n'a pas forcément de connaissances préalables dans le domaine métier. La métaphore devient alors un outil de vulgarisation aussi précieux qu'une « **Conception Simple** »

Paul et Mick dans l'eXtrême

La métaphore est assurément la pratique XP la plus controversée, même entre praticiens.

Elle est donc la moins répandue de toutes les pratiques.

Certains praticiens sont convaincus qu'il faut trouver l'analogie dans le même domaine métier...

D'autres doutent de l'intérêt véritable de cette pratique.

Par expérience, la métaphore est souvent le recours d'un interlocuteur qui souhaite faire passer un message. N'avez-vous jamais discuter avec une personne qui a pris un exemple, une "image" pour se faire comprendre autrement ?!...

Pour ma part, je vous retourne la question : en lisant le précédent chapitre, sur les livraisons fréquentes, avez vous fait attentions aux métaphores ?... Avec le recul, pensez-vous qu'elles vous ont aidé à mieux appréhender le connaissance que je voulais transmettre ?...

Le Planning Game

Au cours du « Planning Game », le client vient avec un ensemble de scénarios utilisateurs suffisant pour couvrir, selon lui, plus d'une itération. Il expose ces scénarios aux développeurs.

Les développeurs posent les questions nécessaires pour s'appropriier ces scénarios.

Ils découpent alors chaque scénario en tâches qu'ils identifient puis estiment consensuellement. La durée d'une tâche ne devant pas dépasser 4 jours. Les développeurs doivent profiter de ce moment privilégié pour confronter et homogénéiser leur connaissance du métier et de l'application.


En sommant l'estimation de chacune des tâches composant un scénario on obtient la durée de chaque scénario.

Le client fera alors le tri parmi les scénarios afin de sortir de l'itération les moins importants selon lui, et de donner une priorité à ceux restant.

Dans le cas d'une itération qui fait suite à une livraison en recette, on pourra éventuellement inclure des tâches de "Traitement des retours de recette éventuels" d'une durée déterminée.

Contrat Creative Commons

Cette création est mise à disposition selon le « Contrat Paternité-Pas d'Utilisation Commerciale-Partage des Conditions Initiales à l'Identique 2.0 France » disponible en ligne <http://creativecommons.org/licenses/by-nc-sa/2.0/fr/> ou par courrier postal à Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.



COMMONS DEED


Paternité - Pas d'Utilisation Commerciale - Partage des Conditions Initiales à l'Identique 2.0 France


Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public
- de modifier cette création

Selon les conditions suivantes :

 **Paternité.** Vous devez citer le nom de l'auteur original.

 **Pas d'Utilisation Commerciale.** Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

 **Partage des Conditions Initiales à l'Identique.** Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

- A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)

Ceci est le Résumé Explicatif du [Code Juridique \(la version intégrale du contrat\)](#).

[Avertissement](#) 