Processus de développement

L'Extreme Programming

Régis Medina

Ed. 2 - 12/2004

TABLE DES MATIÈRES

INTRODUCTION

L'Extreme Programming (XP) est un processus de développement logiciel, c'est-à-dire un ensemble de pratiques destinées à organiser le travail d'une équipe de développement. Ces pratiques se focalisent sur la construction proprement dite du logiciel, en aval des phases préparatoires d'études d'opportunité ou de faisabilité.

XP est l'un des principaux représentants d'une famille émergente de processus : les processus dits "agiles", qui se démarquent des démarches traditionnelles en mettant l'accent sur le travail d'équipe et la réactivité. L'heure est à l'économie et à l'efficacité : « Quelles activités pouvons-nous abandonner tout en produisant des logiciels de qualité ? ». Ou encore : « Comment mieux travailler avec le client pour nous focaliser sur ses besoins les plus prioritaires et être aussi réactifs que possible ? »

XP propose une réponse originale à ces questions, avec un ensemble de pratiques organisées autour des principes suivants :

- Le client (maîtrise d'ouvrage) pilote lui-même le projet, et ce de très près grâce à des cycles itératifs extrêmement courts (1 ou 2 semaines).
- L'équipe livre très tôt dans le projet une première version du logiciel, et les livraisons de nouvelles versions s'enchaînent ensuite à un rythme soutenu pour obtenir un feedback maximal sur l'avancement des développements.
- L'équipe s'organise elle-même pour atteindre ses objectifs, en favorisant une collaboration maximale entre ses membres.
- L'équipe met en place des tests automatiques pour toutes les fonctionnalités qu'elle développe, ce qui garantit au produit un niveau de robustesse très élevé.
- Les développeurs améliorent sans cesse la structure interne du logiciel pour que les évolutions y restent faciles et rapides.

L'objectif de ce dossier est de donner une vision d'ensemble des principes et des pratiques d'XP, en commençant par ce qui fait son agilité : l'ouverture au changement.

OUVERTURE AU CHANGEMENT

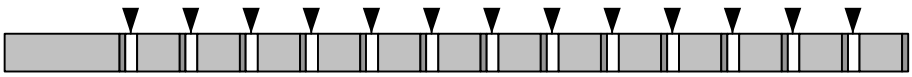
Les démarches traditionnelles, basées sur la fameuse séquence « spécification > conception > réalisation > validation », concentrent la plupart des décisions en début de projet :

L'objectif de cette approche est louable : le client veut des garanties sur ce qu'il obtiendra en fin de projet, et le chef de projet souhaite disposer des informations nécessaires à l'organisation de son équipe.

Malheureusement, les équipes qui évoluent dans un environnement changeant ou complexe savent à quel point il est difficile de s'en tenir aux décisions initiales. Le client réalise que ses besoins ont changé, ou bien l'équipe découvre en phase d'implémentation des erreurs de spécification ou de conception qui compromettent les plans de développement. Le changement s'impose donc tôt ou tard, mais voilà : cette organisation suppose l'absence de changement, et celui-ci se révèle bien vite très coûteux - suffisamment parfois pour compromettre la rentabilité du projet.

Mais puisque le changement est une composante incontournable de tout projet de développement logiciel, pourquoi ne pas l'accepter ? N'existe-t-il pas un moyen pour que les équipes de développement n'opposent plus de rigidité excessive aux demandes de leur maîtrise d'ouvrage ?

Les créateurs d'XP ont trouvé une réponse à ces questions en découvrant que certaines pratiques d'organisation d'équipe et de programmation, appliquées ensemble, permettent de rendre le logiciel extrêmement malléable – à tel point qu'il devient plus avantageux de le faire évoluer progressivement que de chercher à le spécifier et le concevoir complètement dès le départ. Partant de ce constat, ils ont conçu une démarche qui diffuse le processus de décision tout au long du projet grâce à l'enchaînement de cycles itératifs très courts :



Le grand gagnant de cette démarche est d'abord le client du projet. Plutôt que de voir son intervention cantonnée à la phase initiale de recueil du besoin, il intègre véritablement le projet pour en devenir le pilote. A chaque itération, il choisit lui-même les fonctionnalités à implémenter, collabore avec l'équipe pour définir ses besoins dans le détail, et reçoit une nouvelle version du logiciel qui intègre les évolutions en question.

Cette démarche présente de nombreux avantages en termes de conduite de projet :

- Le client jouit d'une très grande visibilité sur l'avancement des développements.

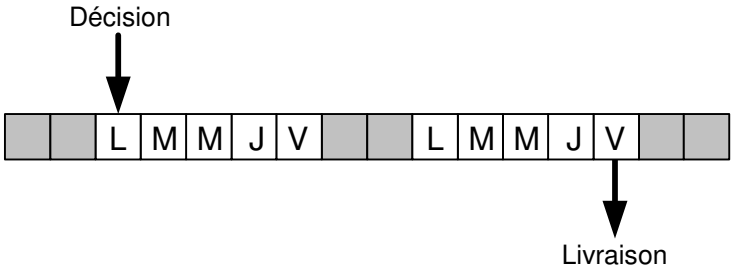
- Le client utilise le logiciel lui-même comme support de réflexion pour le choix des fonctionnalités à implémenter – il peut en particulier intégrer très tôt les retours des utilisateurs pour orienter les développements en conséquence.
- La première mise en production du logiciel intervient très tôt dans le projet, ce qui avance d'autant le moment à partir duquel le client peut en tirer des bénéfices.
- L'ordre d'implémentation des fonctionnalités n'est pas guidée par des contraintes techniques, mais par les demandes du client. Celui-ci peut donc focaliser les efforts de l'équipe sur les fonctionnalités les plus importantes dès le début du projet, et ainsi optimiser l'utilisation de son budget.

La démarche itérative d'XP est donc son principal atout du point de vue de la maîtrise d'ouvrage. Elle est présentée plus en détail dans les sections qui suivent.

LE CYCLE DE DÉVELOPPEMENT XP

L'Extreme Programming vise une réduction significative de la durée du cycle de développement, c'est-à-dire du temps qui s'écoule entre le moment où l'on décide d'implémenter une fonctionnalité et celui où l'on met en production une nouvelle version du logiciel qui intègre la fonctionnalité en question.

Dans un projet XP, ce temps correspond exactement à la durée d'une *itération*, c'est-à-dire typiquement deux semaines. Chaque itération reprend la structure suivante :



- Le premier jour de l'itération est consacré à la *réunion de planification*, au cours de laquelle le client et l'équipe conviennent de ce qui doit être implémenté au cours de l'itération. A la fin de cette journée, l'équipe dispose de la liste précise des tâches à réaliser.
- Ensuite, l'équipe s'organise pour réaliser les tâches en question. Elle prend en charge le suivi des tâches, ainsi que les activités d'analyse du besoin, de conception, d'implémentation et de test correspondantes. Il est important de noter qu'il n'y a pas de changement de cap intermédiaire : l'équipe se focalise sur son objectif sans interruption jusqu'à la fin de l'itération.
- Au terme des deux semaines, l'équipe met une nouvelle version du logiciel à disposition du client. Ce logiciel est robuste, testé, et sa structure interne est laissée aussi propre que possible pour que les prochaines évolutions y restent peu coûteuses.

Cette organisation extrêmement réactive impose de nouvelles contraintes sur le fonctionnement de l'équipe, susceptibles de mettre en défaut les pratiques traditionnelles du développement logiciel. Par exemple :

- Puisque le contenu fonctionnel du produit ne se décide précisément qu'au fur et à mesure des itérations, il n'est plus judicieux de faire reposer l'implémentation sur une conception définie en début de projet. Dans une démarche purement itérative, l'équipe doit être capable de faire émerger la conception tout au long du développement pour suivre les directions données par le client.
- En termes d'organisation d'équipe, il n'est plus judicieux de séparer l'application en modules réservés à tel ou tel développeur puisque pour une itération donnée rien n'empêche le client de demander des fonctionnalités centrées sur un seul module. Tous les développeurs doivent donc être en mesure de travailler sur toute l'application.

XP apporte des solutions concrètes à ces problématiques, avec un ensemble de pratiques qui forme un système cohérent et efficace.

LES PRATIQUES XP

Les pratiques XP sont pour la plupart des pratiques de bon sens utilisées par des développeurs et des chefs de projets expérimentés. On retrouve par exemple les tests unitaires automatisés, les livraisons fréquentes ou encore les relectures de code.

La première nouveauté d'XP consiste à pousser ces pratiques à l'extrême (d'où le nom de la méthode), ou comme le disent ses auteurs "tourner tous les boutons jusqu'à 10 !" L'équipe utilise des cycles de développement d'une ou deux semaines, les développeurs écrivent des tests unitaires pour chaque classe, ils se livrent à une relecture de code permanente via le travail en binômes, etc.

La seconde nouveauté d'XP consiste à organiser ces pratiques en un tout cohérent, de sorte que chaque pratique renforce les autres. Il en résulte une méthode complète, qui couvre tous les aspects du développement - de la relation avec le client jusqu'à l'écriture du code, en passant par les plannings et l'organisation de l'équipe.

Voici les principaux éléments du fonctionnement d'XP :

- **Cycles itératifs pilotés par le client** : Le projet progresse au rythme d'itérations très courtes, dont le contenu fonctionnel est déterminé par le client.
- **Travail d'équipe auto-organisé** : L'équipe travaille réellement... en équipe. Les développeurs organisent eux-mêmes leur travail, interviennent sur l'ensemble du code, travaillent systématiquement en binômes, et synchronisent leurs développements plusieurs fois par jour.
- **Programmation pilotée par les tests** : les développeurs écrivent des test automatiques pour chaque portion de code qu'ils conçoivent, et ils s'appuient sur ces tests pour affiner et améliorer sans cesse la conception de l'application sans craindre de régression.

Pilotage du projet

Le rôle de « client XP »

Le pilotage du projet est assuré par un membre spécifique de l'équipe projet : le « client ». Le client détermine les fonctionnalités du logiciel, gère les priorités, définit les spécifications précises du produit – en somme, ce rôle correspond à ce que nous nommons en France la maîtrise d'ouvrage du projet. Dans un projet XP, ce représentant de la maîtrise d'ouvrage rejoint le projet à plein temps.

Le choix de cette personne dépend très fortement du type de projet. Dans les contextes les plus simples, il pourra s'agir d'une personne qui est à la fois donneur d'ordre et utilisateur final. Dans d'autres contextes, le client pourra être le représentant d'un groupe plus large réunissant un comité de direction, des architectes, etc.

La phase initiale d'exploration

Le projet démarre par une phase d'exploration volontairement très courte (typiquement un mois maximum), qui a pour triple objectif de définir le contenu fonctionnel de l'application, établir un premier plan de développement pour le projet, et produire la toute première version du logiciel.

Au cours de cette phase, le client explore et définit le contenu fonctionnel qu'il souhaite voir implémenté dans le produit. Il établit ainsi une liste de fonctionnalités, qui prennent en XP la forme de « scénarios client ».

Un scénario client décrit en quelques mots un besoin particulier du client. Par exemple, pour un logiciel de gestion de carnet d'adresses le client pourrait écrire les scénarios suivants :

- "Je rentre un nom ou prénom, et le logiciel affiche la liste de toutes les personnes qui possèdent ce nom ou ce prénom"
- "Je peux exporter mon carnet d'adresses au format HTML."

Les scénarios sont rédigés dans le langage du client, et décrivent des fonctionnalités dont l'implémentation paraît assez rapide – un scénario doit en effet pouvoir être complètement implémenté en une itération. Si un scénario paraît trop vague ou trop complexe, il est décomposé en scénarios plus simples. A l'inverse, des scénarios trop courts peuvent être regroupés en un seul pour obtenir la granularité souhaitée.

Au cours de la phase d'exploration, le client et l'équipe se forgent ainsi une idée assez précise du périmètre fonctionnel souhaité de l'outil. Ils établissent alors le premier plan de développement du projet.

Planification du projet

La planification est réalisée au cours d'une réunion dédiée, qui réunit l'équipe et le client et se déroule comme suit :

1. Le client présente les différents scénarios à l'équipe, qui tente de se faire une idée de la charge de travail de chacun d'entre eux.

2. L'équipe donne pour chaque scénario une estimation de son coût d'implémentation, en « points » abstraits : tel scénario coûte 3 points, tel autre 1 point, etc.
3. L'équipe donne au client une estimation de sa « vélocité », c'est-à-dire du nombre de points de scénarios qu'elle s'estime capable de traiter en une itération – par exemple 10 points.
   Au tout début du projet cette vélocité est seulement estimée, mais après les premières itérations elle est réajustée en adoptant la règle suivante : la vélocité estimée pour une itération donnée correspond exactement au nombre de points effectivement traités à l'itération précédente.

   > ***Remarque***
   > *La vélocité ne représente en aucun cas le niveau de "performance" de l'équipe, dans la mesure où elle dépend en grande partie de la façon dont les développeurs font leurs estimations. Cependant, ses variations peuvent indiquer des problèmes passagers : si la vélocité baisse brutalement, c'est peut-être que l'équipe a été ralentie pour une raison ou une autre, et il ne faut pas ignorer cette information.*
4. Le client établit lui-même le plan de développement en affectant les différents scénarios aux itérations à venir, de sorte qu'à chaque itération la somme du nombre de points des scénarios choisis soit égale à la vélocité annoncée.

Côté outillage, on donne la priorité à l'aspect participatif de la démarche en notant les scénarios sur des fiches cartonnées que l'on colle sur un grand tableau ou un mur :

Cette pratique peut surprendre dans des contextes où les documents ont souvent une forme électronique, mais les séances de planification jouent avant tout sur l'aspect humain du projet, la communication directe entre l'équipe et le client permettant d'aligner les deux parties sur l'objectif à atteindre.

Le plan de développement ainsi constitué est disposé à proximité de l'endroit où travaille l'équipe, de manière à ce que celle-ci ait une vision toujours à jour de ses engagements.

Première mise en production

Comme nous l'avons évoqué plus haut, la phase d'exploration se termine par la première livraison du logiciel.

Cette première mise en production intervient aussi tôt que possible dans le projet : il faut trouver le contenu fonctionnel minimal qui commence à avoir un sens pour les utilisateurs, quitte à faire preuve d'imagination en amenant par exemple le nouveau logiciel en complément d'un système existant dans le cadre d'une fonctionnalité bien précise.

Livraisons suivantes

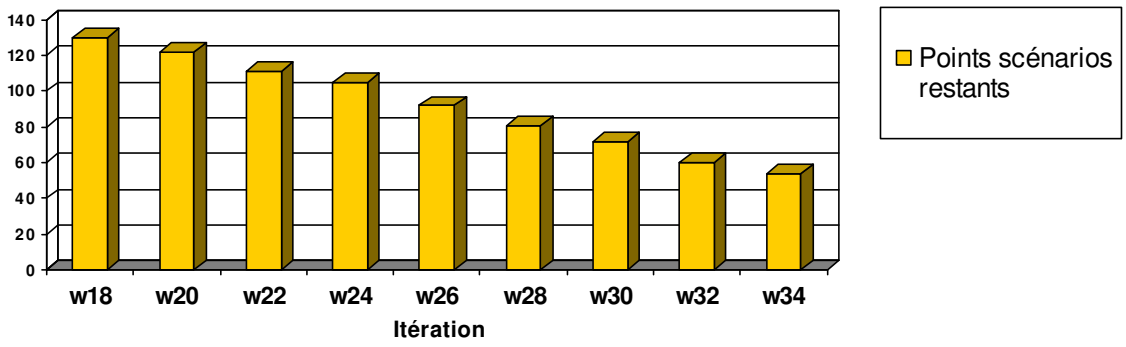
Les mises en production s'enchaînent ensuite à un rythme régulier, toujours fixé par le client. L'objectif est d'obtenir un feedback très rapide sur le développement – en pratique toutes les une à six itérations selon la complexité de déploiement du produit.

Le plan de développement est continuellement mis à jour si nécessaire pour tenir compte des événements suivants :

- L'équipe de développement progresse à une vitesse différente de celle prévue (dans le bon ou le mauvais sens)
- Le client décide de changer le contenu des itérations restantes. Il peut ainsi permuter des scénarios en fonction de nouvelles priorités, ou encore remplacer certains scénarios du plan par de nouveaux.

Suivi du projet

Le suivi « haut niveau » du projet peut par exemple s'appuyer sur des graphes inspirés de celui représenté ci-dessous, dans lequel on note le nombre de points de scénarios restant à implémenter.



Tous les scénarios imaginés par le client ne sont pas nécessairement représentés dans ce graphe. Seuls y figurent ceux qui ont été retenus pour la prochaine grande échéance – typiquement la prochaine version majeure du produit, ou encore la fin du projet pour un développement de type forfait.

Cette courbe permet par projection d'estimer si les objectifs du projet seront tenus, et elle permet aussi d'identifier certains « patterns » dans le fonctionnement de l'équipe[1] - par exemple une tendance à la sur- ou sous-estimation des tâches.

---

[1] Le livre « Agile Software Development with Scrum » de Ken Schwaber et Mike Beedle comporte une très intéressante présentation de ce type d'analyse.

Tests de recette automatiques

Pour chaque scénario planifié, un ensemble de tests de recette est écrit. Ces tests ont pour but de vérifier de manière automatique (c'est-à-dire sans intervention ou interprétation humaine) chacune des fonctionnalités demandées par le client. Le client définit ces tests et participe éventuellement à leur implémentation, assisté pour cela d'un certain nombre de testeurs.

Ces tests peuvent prendre plusieurs formes. Il peut s'agir de jeux de données, sous forme par exemple de feuilles de tableur ou de fichiers XML, qui définissent une transformation effectuée par le logiciel : « pour telle entrée, le logiciel doit produire tel résultat ». Il peut s'agir également de scripts pour les cas plus complexes, ces scripts décrivant par exemple des séquences d'interactions de l'utilisateur avec l'interface graphique du produit.

Ces tests représentent dans un projet XP les spécifications détaillées de l'outil – des spécifications formelles, toujours en phase avec le développement. Dans un contexte « pur XP », ce sont les seules spécifications : il n'y a pas de *document* de spécifications à proprement parler. En pratique, cependant, des documents peuvent être exigés par l'organisation qui encadre le projet. Des documents synthétiques sont alors réalisés par le client.

Les tests de recette sont lancés fréquemment, et tous les participants du projet (client, direction, développeurs) sont informés en permanence de leurs résultats[2].

[2] Des outils d'intégration continue tels que « Cruise Control » ou « Damage Control » peuvent être utilisés à cette fin.

Travail d'équipe auto-organisé

L'organisation d'une équipe XP s'éloigne de l'organisation traditionnelle d'une équipe de développement, dans lequel différents développeurs travaillent sur des parties distinctes de l'application, coordonnés par un chef de projet responsable de l'intégrité de l'ensemble. Ce schéma de *séparation* des activités fait place dans XP à un schéma d'*intégration*, ou de collaboration intensive.

Définition et partage des tâches

Au début de chaque itération, le client, les développeurs et les testeurs se réunissent au cours d'une « réunion de planification d'itération ». La réunion commence par un point sur l'itération passée, puis le client décrit les scénarios à implémenter au cours de l'itération qui commence. S'ensuit une discussion sur les scénarios en question, au cours de laquelle l'équipe dresse une liste des tâches correspondantes.

Cette réunion se présente comme une réelle séance de conception collective, qui donne aux différents intervenants l'occasion d'aligner leur compréhension de ce qui doit être réalisé. Les besoins sont analysés à un niveau de détail plus fin, et les développeurs commencent à envisager les solutions techniques à mettre en place.

Au terme de cette réunion, l'équipe dispose d'une liste complète des tâches à réaliser dans l'itération. L'attribution des tâches se fait alors de manière auto-organisée : les développeurs choisissent eux-mêmes leurs tâches en cours d'itération, ce qui est rendu possible par deux pratiques fondamentales d'XP : la responsabilité collective du code, et le travail en binômes.

Responsabilité collective du code

Chaque développeur d'une équipe XP est susceptible de travailler sur toutes les parties de l'application, sans être bloqué par une éventuelle spécialisation initiale. Cette liberté d'action s'accompagne toutefois d'une responsabilité : chaque membre de l'équipe est garant de la qualité de l'ensemble de l'application – en particulier, chacun a le devoir de laisser le code qu'il parcourt aussi clair et propre que possible, pour faciliter le travail de ceux qui y interviendront par la suite.

Travail en binômes

Les développeurs d'une équipe XP travaillent quasiment tout le temps en binômes, c'est-à-dire à deux sur un même poste. L'idée n'est en aucun cas d'avoir une personne qui code pendant que l'autre la surveille, mais au contraire d'aboutir à un dialogue permanent par lequel les deux membres du binôme sont engagés à 100% dans leur tâche.

Les binômes changent fréquemment (typiquement tous les jours), de sorte que chacun est rapidement amené à travailler avec tout le reste de l'équipe. C'est cela qui rend possible la pratique de responsabilité collective du code : un développeur donné peut intervenir sur une partie de l'application qu'il ne connaît pas encore bien, sachant qu'il pourra s'associer à un binôme ayant déjà travaillé sur le sujet. Après quelque temps de ce traitement, chacun a une bonne vision d'ensemble de l'application et peut intervenir n'importe où.

Cette mécanique de rotation des binômes et de responsabilité collective du code conduit à une très grande souplesse en matière d'organisation d'équipe : le projet n'est plus bloqué par

l'absence de tel ou tel individu, et toute l'équipe peut focaliser ses efforts sur une partie donnée de l'application en cas de besoin.

Stand-up meetings

Chaque jour, toute l'équipe se réunit pour un « stand-up meeting » d'une quinzaine de minutes - les participants se tiennent d'ailleurs tous debout (d'où le nom) pour éviter que cela ne traîne. A l'occasion de cette réunion, tous les membres de l'équipe prennent la parole pour faire un point sur l'avancement de l'itération, et surtout pour organiser la journée de travail à venir.

Règles de codage et métaphore

Des règles de codage sont définies et suivies par l'ensemble des développeurs. Elles donnent au code un aspect uniforme sur toute l'application, ce qui facilite le partage du code et le travail en binômes.

Côté conception, l'équipe vise l'intégrité d'ensemble en s'appuyant sur un système de métaphores commun, c'est à dire un ensemble d'expressions décrivant les acteurs du système et leurs interactions. Ce système de métaphores constitue :

- Un vocabulaire commun qui va permettre à toute l'équipe de parler des mêmes choses avec les mêmes mots. Par exemple, dans le monde des télécom, il faut définir ce qu'est un "noeud" d'un réseau et se servir de ce mot dans le cadre de la définition établie.
- Une métaphore de fonctionnement, par exemple : "le logiciel de contrôle de cette machine-outil fonctionne comme une machine à café".

Intégration continue

L'intégration des modifications menées par les développeurs est faite tous les jours. Dès qu'un binôme finit une tâche, il met à jour la version d'intégration en s'assurant que tous les tests de non-régression de l'application passent à 100%. La version "livrable" du logiciel évolue donc chaque jour, reflétant fidèlement l'état d'avancement des développements.

Le rôle du coach

Dans un contexte où l'équipe s'organise elle-même en définissant et en choisissant ses tâches, le rôle de chef de projet tel que nous le connaissons devient inadéquat. L'activité de coordination de l'équipe est prise en charge par un « coach », qui est davantage un facilitateur qu'un donneur d'ordres. Le but du coach est en effet de former l'équipe aux pratiques XP, et ensuite de travailler en permanence sur les pratiques de l'équipe pour améliorer son fonctionnement et l'amener à l'autonomie.

Les fonctions hiérarchiques et administratives du chef de projet font l'objet dans XP d'un rôle particulier : le « manager ». Le manager est le patron de l'équipe, il s'assure qu'elle dispose des moyens est nécessaire à son fonctionnement, fait l'interface avec le reste de l'organisation, et vérifie enfin que les résultats sont bien là.

Rythme durable

D'une part, une équipe surmenée ne produit pas un bon travail. D'autre part, s'il y a trop de retard, c'est qu'il y a un problème de fond et il est illusoire de chercher à le masquer par davantage de travail. Pour ces raisons, l'équipe adopte la règle suivante : pas d'heures supplémentaires deux semaines de suite.

PROGRAMMATION PILOTÉE PAR LES TESTS

La démarche extrêmement itérative proposée par XP n'est viable que si l'équipe est en mesure de garder la maîtrise technique de l'application, c'est-à-dire de faire en sorte que les modifications du code restent longtemps faciles et rapides malgré les nombreuses évolutions. Cette maîtrise n'est pas le fruit du hasard, elle est le résultat d'une discipline de développement qui s'appuie sur les pratiques suivantes :

Mise en place intensive de tests unitaires

En complément des tests de recette, qui servent à prouver au client que le logiciel remplit ses objectifs, XP utilise intensivement les tests unitaires de non-régression. Ces tests sont écrits par les développeurs en même temps que le code lui-même, pour spécifier et valider le comportement de chaque portion de code ajoutée.

Quasiment chaque classe de l'application possède une classe jumelle de test. Lorsque des développeurs doivent ajouter une nouvelle méthode à la classe applicative, ils commencent par ajouter à la classe de test un ensemble d'assertions qui décrivent le comportement de la méthode à ajouter. La méthode n'est implémentée qu'ensuite, pour que ces tests passent. Il est important de noter que les tests s'exécutent sans aucune interprétation de la part du développeur : les assertions en question doivent vérifier automatiquement si le code testé fonctionne ou non.

Les classes de tests ne sont pas jetées après usage : elles sont gérées par un framework dédié[3] qui permet de les exécuter individuellement ou en totalité. La batterie de tests de l'application ne cesse donc de s'enrichir, et forme avec le temps un filet de sécurité qui permet aux développeurs d'effectuer des modifications dans le code existant sans crainte de régression.

Ces tests sont lancés à longueur de journée par les développeurs - en fait, à chaque compilation. Tous les tests unitaires doivent passer à 100% dans l'environnement d'un binôme avant tout report de modifications dans la version d'intégration du logiciel.

Cette pratique constitue l'une des pratiques centrales de XP. Ses avantages sont tels que chaque équipe, XP ou non, se doit de s'y pencher avec la plus grande attention :

- Les erreurs sont très vite détectées et localisées.
- Les tests donnent une vision précise de la tâche à réaliser et aident les développeurs à aller droit au but.
- Les tests placent les développeurs dans un contexte réduit qui leur permet de s'abstraire de la complexité du reste de l'application.
- Il est possible de s'assurer du fonctionnement global du système très rapidement, après toute modification du code, avant ou après toute intégration dans la version officielle du système.

Ce que l'on constate au final, c'est que les développeurs d'une équipe XP passent plus de temps à ajouter des fonctionnalités, et moins de temps à investiguer des défauts dans le code existant.

---

[3] Par exemple JUnit : http://www.junit.org

Remaniement du code

L'application est toujours maintenue aussi malléable que possible grâce à une activité permanente de remaniement (*refactoring*). Le remaniement consiste à retoucher ou réorganiser des portions de code existantes, à comportement fonctionnel identique, en vue d'améliorer la structure d'ensemble de l'application.

L'une des motivations principales de remaniement est l'absence de duplication dans le code : tout doit y apparaître "Once and Only Once". Lorsqu'un développeur doit coder une nouvelle fonction proche d'une fonction déjà existante dans le code, il remanie l'application de manière à pouvoir utiliser directement le code existant. De la sorte les modifications de l'application n'impactent généralement que peu d'endroits dans le code, ce qui allège d'autant le travail.

Le remaniement est également utilisé pour rendre le code plus clair : lorsqu'un développeur intervient sur une partie donnée du code, il la modifie si nécessaire afin de faciliter le travail de ceux qui passeront après lui.

Le remaniement est plus qu'un moyen de dépoussiérer le code : c'est une façon de faire émerger une conception aussi adaptée que possible aux besoins de l'application, en supprimant au fur et à mesure tout ce qui nuit à sa simplicité et peut ralentir l'équipe. Comme les tests unitaires, le remaniement est pratiqué à longueur de journée - il s'agit d'une discipline quotidienne de qualité, et non d'une activité épisodique de réécriture.

Conception simple

"*You ain't gonna need it!*" : on ne fait de conception que pour les fonctionnalités existantes, pas pour les fonctionnalités futures. En corollaire :

- On ne fait de généralisations dans la conception que lorsque des besoins concrets se font sentir.
- On n'introduit pas d'optimisations si elles ne sont pas demandées par le client.

En somme, plutôt que de préparer le logiciel pour un futur hypothétique, XP préconise de s'assurer que le travail actuel soit toujours bien fait (code testé, simple, lisible et sans duplications) de sorte que les changements restent faciles et rapides le moment venu.

Contrairement à ce que l'on pourrait penser au premier abord, cette position n'est pas celle de développeurs désireux de se consacrer à l'implémentation sans perdre de temps à concevoir. Il s'agit plutôt de l'inverse : en s'interdisant d'avoir un code médiocre, une équipe XP fait de la conception une activité vitale pour son fonctionnement. On observe d'ailleurs en pratique que les développeurs d'une équipe XP passent la plupart de leur temps à travailler sur cette conception.

LES VALEURS D'XP

Les processus agiles tendent à s'éloigner d'une approche Taylorienne, à la fois « scientifique » et impersonnelle, du travail. Ils mettent au contraire l'accent sur l'élement humain, et parient sur la performance d'une équipe dont les membres travaillent en collaboration étroite – l'équipe en question incluant notamment la maîtrise d'ouvrage du projet.

Plus généralement, les pratiques XP sont sous-tendues par les quatre valeurs suivantes :

- **Communication** : XP favorise le contact humain, la communication directe, plutôt que le cloisonnement des activités et les échanges de courriers électroniques ou de documents formels. Les développeurs travaillent directement avec la maîtrise d'ouvrage, les testeurs sont intégrés à l'équipe de développement, etc.

- **Feedback** : qu'il s'agisse d'itérations courtes, de livraisons fréquentes, de travail en binômes ou de tests automatiques éxécutés en permanence, la plupart des pratiques XP sont conçues pour donner un maximum de feedback sur le déroulement du projet afin de corriger la trajectoire au plus tôt. En particulier, les points de début d'itération offrent à l'équipe le moyen de prendre du recul sur son fonctionnement et de l'améliorer sans cesse au fil des itérations.

- **Simplicité** : comme nous l'indiquions au début de ce dossier, XP relève le défi suivant : « que pouvons-nous arrêter de faire tout en continuant à créer efficacement un logiciel qui réponde aux besoins réels du client ? ». Cette recherche de simplification touche le processus lui-même, mais aussi l'outil fabriqué (la mécanique de planification incite le client à focaliser les efforts sur les fonctions prioritaires) ou encore la conception de l'application (guidée par un principe de « You ain't gonna need it »).

- **Courage** : il s'agit principalement du courage d'honorer les autres valeurs – celui de maintenir une communication franche et ouverte, ou encore d'accepter et de traiter de front les mauvaises nouvelles.

C'est en définitive sur ces critères qu'une équipe XP juge de sa maturité et trouve les principaux axes de son amélioration.

CONCLUSION

Nous mettons en œuvre ces pratiques XP depuis bientôt trois ans dans le cadre d'un grand projet télécom[4], et les bénéfices de cette démarche nous semblent indéniables :

- L'approche itérative nous a permis d'améliorer progressivement le produit dans son ensemble, et nous recevons aujourd'hui d'excellents retours des clients finaux.
- Les développeurs apprécient cette démarche, et les équipes sont jugées très réactives par leurs interlocuteurs.
- Nous disposons aujourd'hui d'une batterie de plus de 11.000 tests, grâce auxquels le nombre de « bugs » du produit est resté très bas.
- Notre activité d'amélioration permanente de la conception nous a permis de positionner une partie de l'outil en tant que framework utilisé par d'autres équipes de la société.

Bien entendu, tout n'est pas complètement rose. Nous mettons en oeuvre l'ensemble des pratiques « internes » d'XP, celles qui concernent le travail de l'équipe, mais nous avons des difficultés à progresser sur les parties liées au rôle du client et du testeur – et pour cause, ces rôles sont pris en charges par d'autres équipes, ce qui exige de notre part un effort d'évangélisation de longue haleine.

Nous avons pu également noter la grande sensibilité de la méthode à l'entente des membres de l'équipe. Cela impose des contraintes sur le recrutement, et cela peut également nécessiter des remaniements de l'équipe en cas de dysfonctionnement.

D'une manière générale, nous avons constaté que la mise en œuvre d'XP apportait des résultats indéniables, mais au prix d'efforts eux aussi indéniables lors de la phase de transition. Une fois la transition effectuée, cependant, XP se présente comme un processus simple et naturel, qui offre une productivité nettement accrue par rapport à ce que nous avions connu jusque là.

Cette difficulté de transition explique certainement pourquoi XP reste encore confidentiel en France. Nous avons eu ainsi des retours d'équipes ayant essayé « XP mais sans les tests unitaires », ou encore « XP mais sans le travail en binômes », avant de revenir à leur mode de fonctionnement initial quelques semaines plus tard.

Il faudra donc une impulsion particulière pour que XP se développe davantage dans notre pays. Nous pensons que cette impulsion viendra de deux directions complémentaires : d'abord des developpeurs eux-mêmes, qui auront compris en quoi XP constitue une façon plus naturelle et plus efficace de travailler, et ensuite des maîtrises d'oeuvre, maîtrises d'ouvrage et sociétés de service « visionnaires » qui auront réalisé que ce processus représente aujourd'hui un réel avantage compétitif dans un monde où l'adaptation au changement est un facteur primordial de succès.

[4] Voir à ce propos nos deux dossiers « Retour d'expérience XP ».