INGÉNIEURS 2000

Filière Informatique et Réseaux
3ème année

Exposé de système

eXtreme Programming

Adrien Machado

Professeur responsable
M. Dominique Revuz

Année 2002

Table des matières

I. L'historique

A. Un constat alarmant

La naissance d'une nouvelle méthode arrive rarement sans raisons. Elle fait plutôt suite à une période difficile…

1. Les 3 plaies du développement logiciel

D'après une étude américaine menée en 1994 sur 8000 projets, 16 % n'ont pas respecté les délais et le budget initial et, pire, 32 % n'ont jamais abouti.
Ainsi, dans le monde de l'ingénierie logicielle, trois plaies pourtant bien connues enveniment le développement d'applications.
La première concerne le planning difficilement maîtrisé ou impliquant de nombreuses heures sup. Le deuxième est issue des besoins souvent mal identifiés ou mal compris en raison d'un cahier des charges mal étudié ou incomplet. Tout cela implique des changements au cours du développement, d'autant plus si le client décide de modifier le produit, après une première maquette par exemple. Enfin, le dernier problème concerne la livraison finale du produit qui est parfois buguée.

2. La difficulté de réalisation

Pour cause la conception d'un software n'est pas, contrairement à son nom, souple en raison de la quantité de travail engendrée traditionnellement par une modification, même minime du cahier des charges. Pour exemple, une simple demande exige la remise en question des tests prévus et une modification parfois très importante de l'architecture du logiciel. On considère alors que le coût du changement croit exponentiellement avec le temps.
Avec près de 80% des projets, toutes tailles confondues, qui ne respectent pas les contraintes de départ de temps et de budget, les chefs de projet s'interrogent de plus en plus sur leurs méthodes.
Historiquement, lors du développement d'applications et de projets e-business, chaque développeur se voit confier une partie du travail, le tout étant ensuite assemblé. Or, cette organisation fait perdre du temps, car tous les participants au projet ne travaillent pas de concert. Par la suite, on essaye plusieurs solutions dont les résultats ne sont que rarement concluant. La première est de tenter de rattraper du retard grâce à des heures supplémentaires, mais la qualité en souffre et le moral des participants aussi. La deuxième est de toute façon de se conformer aux objectifs qualités coûte que coûte en tentant de rallonger le délai ou de choisir consciemment de ne pas livrer une partie du projet. Dans ces cas, le client n'est que rarement content… Enfin, une dernière solution est de tenter de requérir plus de ressources mais tous les responsables savent que c'est difficile, et le mot est faible… Ainsi, soit vous ne les obtenez pas, ce qui vous met en mauvaise position face à votre responsable, soit vous les obtenez et les coûts administratifs vous enterrent…
A l'approche des dates butoires, on a donc un certain stress qui amène à prendre certaines décisions qui, le plus souvent, entraînent un mécontentement du client.

B. Le père d'XP : Kent Beck

XP est né lors d'une récente vague de création de nouvelles méthodes parallèlement à l'explosion de l'informatique dans les entreprises et de l'émergence du freeware.
Lors de cette vague, sont essentiellement apparues des méthodes qualifiées d'agile dont XP fait partie.

1. Chrysler, le berceau d'XP

Si l'on recherche plus précisément le berceau de la méthode, on tombe sur un grand projet du groupe Daimler Chrysler datant du milieu des années 90 et nommé le « Chrysler Comprehensive Compensation » ou « *C3* ». Celui-ci consistait à remettre à jour le système de paie de tout le personnel gérant plus de 10 000 personnes. Le logiciel d'origine contenait 2000 classes Smalltalk et 30 000 méthodes dont les modules transmettaient des données faussent, bien que le programme réponde aux spécifications et cela malgré 18 mois de développement et des millions de dollars investis… Quand Kent Beck (un gourou reconnu du monde Smalltalk) repris le projet en 1996

accompagné de Ward Cunningham, ils ont préféré dire à la responsable informatique de Chrysler, Sue Unger, de repartir à zéro plutôt que continuer sur de mauvaises bases. Cette dernière, en concertation avec ses collègues, décida de se lancer dans cette « monstrueuse » aventure. Toutefois, elle confia le projet à Beck qui lui dit pourtant : « Vous ne comprenez pas, je suis consultant, je ne vais pas sur le terrain », auquel elle a répondu : « Vous ne comprenez pas, je m'appelle Sue Unger, et vous irez sur le terrain »…

2. Beck, un chef de projet novateur

Beck sorti alors de sa manche un bon nombre de méthodes qu'il avait auparavant testées mais il se rendit compte qu'aucune lui permettrait de venir à bout de ses contraintes et il devait innover… "C'était un mélange de préparation soigneuse et de panique », disait-il. Tout d'abord, le projet devait se dérouler en petites itérations définies par le client et testées systématiquement pour montrer l'avancement. Le temps étant compté, réaliser les tests avant le code allait permettre de réaliser des économies de temps et de personnel puisque l'équipe « assurance qualité » devenait inutile en raison de la correspondance de la production avec la demande. Souhaitant encore faire des économies de temps, la programmation en tandem fut choisie. En effet, à deux le déboguage est instantané et la rédaction de la documentation est facilitée : « Si un programmeur parvient à communiquer clairement ses idées à un autre programmeur travaillant à ses côtés, il y a de grandes chances pour que celui qui vérifiera le programme deux ans plus tard n'ait aucun mal à le comprendre ».
Au fur et à mesure de l'arrivée de ses idées, Beck devenait persuadé du bien fondé de sa méthode. "Au bout de quinze personnes, c'était devenu tout à fait concevable", se souvient-il. Durant l'été 1996, il décida de baptiser son concept. Puisque les autres méthodes donnaient la priorité au planning sur la programmation, Beck décida que "programmation" devrait faire partie du nom. "J'avais alors besoin d'un adjectif attrayant, suffisamment descriptif et défendable." Il décida alors de nommer sa méthode "extreme programming", après avoir découvert la puissance des athlètes de l'extrême. "Je voulais que mes équipes aient le même sentiment d'invulnérabilité face aux défis à relever." De plus, les pratiques sur lesquelles est bâtie XP sont « autant de boutons de contrôle qu'il pousse au maximum ». Ainsi, cette méthode est la réunion cohérente de pratiques peut-être simples mais portées à un point extrême.
Et finalement, sa méthode fut un succès et en 1997, Chrysler avait un nouveau système de paie, et Beck, assisté de son collègue et ami Ron Jeffries, écrivait le premier ouvrage d'une série sur la "méthode XP" tout en s'appliquant à faire des adeptes
C'est donc ce projet qui est à l'origine d'XP et qui a vérifié et testé ses grands principes, en tirant des enseignements des erreurs de management précédentes.

C. Un développement rapide dans le monde

La méthode est née réellement en 1997 après la parution du premier livre de Kent Beck : « eXtreme Programming Explained ».

1. Un développement international facilité par internet

La toile a alors joué depuis un grand rôle dans le développement de la pratique d'XP. Pour preuve, il existe de nombreux forums de discussion qui lui sont directement consacrés, comme par exemple, pas moins de 71 « Yahoo Groups » internationaux.
Parmi eux, le plus connu et le plus populaire a été créé en décembre 1999 et possède aujourd'hui près de 3400 membres avec une moyenne de 1850 messages postés par mois.
On commence ainsi à trouver XP dans des la plupart des domaines comme l'électronique (développement des pilotes de matériels), la banque, le transport et les télécoms.

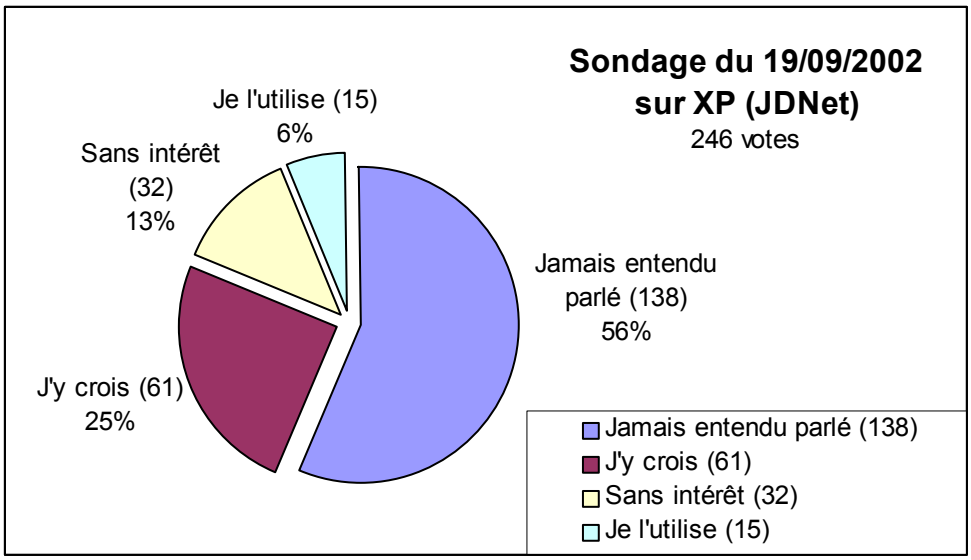
La méthode fait aussi une incursion chez certains éditeurs ou dans les entreprises de secteurs scientifiques et techniques, pour leurs projets de R&D. Quelques sociétés de services, dont des agences web, s'y intéressent également, voyant dans XP un moyen de raccourcir leurs délais. Un auto-recensement international des projets en XP est d'ailleurs fait à cette page :

- http://c2.com/cgi/wiki?ExtremeProgrammingProjects

- Voici quelques exemple :
- Bayerish landes bank, Credit swiss life, First union national bank
- Daimler-Chrysler, Ford Motor Company, BMW
- CSEE transport
- Nortel
- AGF

2. XP s'installe progressivement en France

En France, pour l'heure, les premières mises en oeuvre se limitent essentiellement à des projets réduits ou non stratégiques. La méthode ne semble en effet pas assez connue par les directions. Pour preuve, dans un sondage 19 septembre 2002 du JDNet, plus de la moitié des visiteurs n'ont jamais entendu parlé de la méthode…



Mais la communication va bon train et on voit même apparaître de plus en plus de formations à XP et des cours dans des universités (Université d'Artois, de Genève…) et des grandes écoles, notamment à l'INSA Lyon ou à l'école des mines de Nancy, qui sont tout de même des références dans le monde informatique.
On voit aussi apparaître des sociétés françaises qui se vantent de maîtriser la méthode afin d'attirer le client…
La communauté des XPiens française est bien existante et communique sur son « Yahoo groupe » : extremeprogramming-France comptant environ 250 membres. Les co-auteurs, Laurent Bossavit et Dominic Williams, du livre francophone « **L'eXtreme Programming : Avec deux études de cas »**, participent d'ailleurs activement aux discussions.
XP est donc encore à ses débuts, en terme de connaissance mais est en phase de devenir une référence mondiale. A suivre…

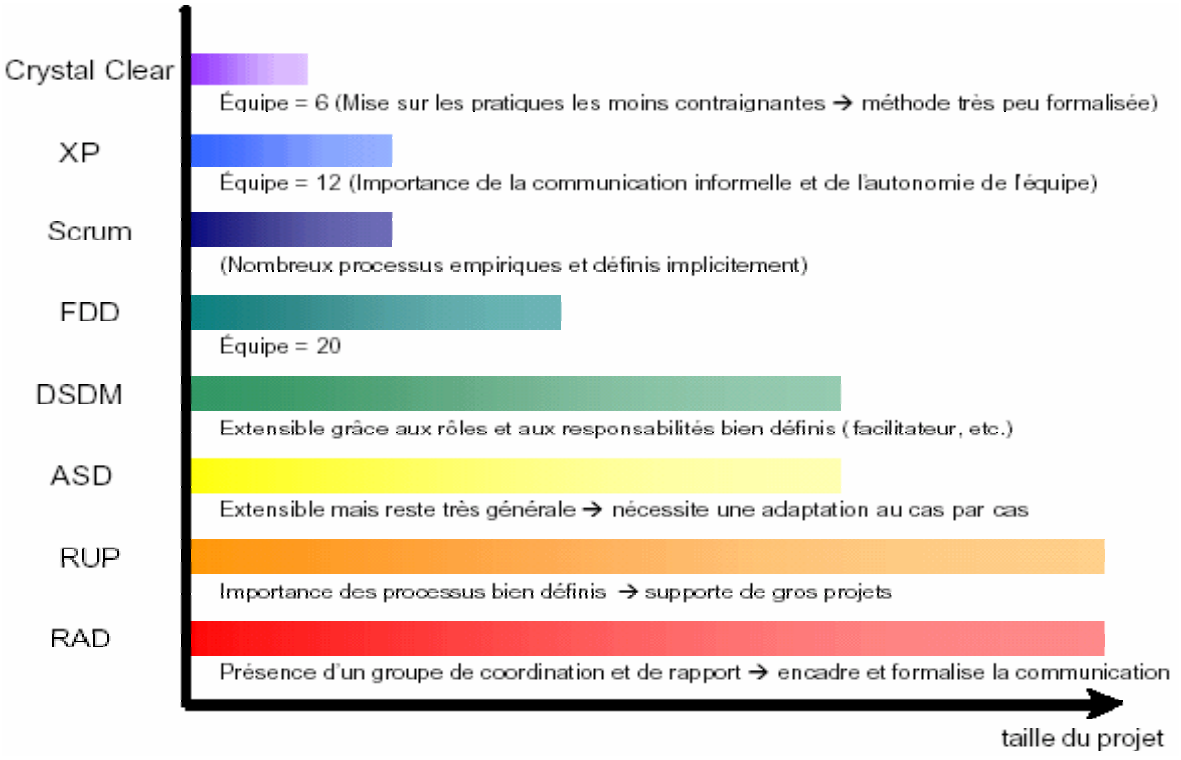
D. Situation par rapport aux autres méthodes

Le but de cette partie est de situer XP par rapport à d'autres méthodes et en aucun cas de présenter d'autres méthodes.
Pour plus d'informations sur d'autres méthodes : chefdeprojet.com et plus particulièrement sur les méthodes agiles : livre blanc

1. XP : une méthode agile

Les méthodes agiles sont des méthodes de gestion de projets informatiques et de développement qui se positionnent en réaction à des méthodes traditionnelles jugées trop lourdes. XP fait partie de cette

catégorie dont la première idée a été de réagir face à la bureaucratie où toute démarche exige une grande quantité de temps. C'est aussi la plus populaire des méthodes agiles mais on peut aussi citer : ASD (Adaptative Software Development), SCRUM, Crystal, FDD (Feature Driven Development) et DSDM (Dynamic System Development Method) qui peuvent chacune être adaptée à une situation particulière, fonction de la taille de l'équipe, comme le montre le graphique ci-après, tiré du site : http://www.rad.fr/



Depuis le mois de février, les figures de proue des méthodologies de développement agiles ont constitué l'Agile Alliance http://www.agilealliance.com.
En avril 2002, le PUMA (Proposition pour l'Unification des Méthodes Agiles) naît poussé par Jean-Pierre Vickoff, consultant formateur, directeur de projet et architecte de SI, auteur de nombreux ouvrages sur les méthodes et expert RAD. Voir http://site.voila.fr/radcp/forumpuma2.PDF pour une présentation.

2. XP vs Méthodes traditionnelles comme UML

Par rapport aux autres méthodes plus traditionnelles comme RAD et Unified Process (qui s'appuie sur UML), Jean-Louis Bénard, Directeur Technique du groupe Business Interactif indique qu'il serait une erreur de les opposer aux méthodes agiles. « Les méthodes agiles essayent de formaliser le bon sens et le pragmatisme que des équipes mettent déjà en oeuvre dans le cadre de projets dits traditionnels ».
D'ailleurs, XP peut être vu comme une déclinaison de Unified Process (UP), en ayant bien en tête cependant que c'est le client qui rédige les spécifications en XP contrairement à UP où les spécifications sont une retranscription par l'informaticien de ce qui a été compris. Aussi, les itérations qui sont un des principes des deux méthodes sont environ 6 fois plus fréquentes en XP.
Ainsi, les écoles classiques et « agiles » sont complémentaires.

UML qui est une méthode de modélisation n'est par essence pas « menacé » par l'engouement pour XP qui est une méthode de réalisation, d'organisation.
D'ailleurs, XP utilise le langage universel qu'est UML pour communiquer. Toutefois, par sa volonté de rapidité, XP reste prudent sur l'utilisation d'UML. Pour cause, les diagrammes UML sont trop longs à réaliser alors que la conception en XP étant très courte et réalisée au fur et à mesure. La

documentation par les diagrammes UML est aussi déconseillée en raison du volume de ces derniers, par rapport aux quelques pages préconisées dans XP.

II. Les fondements D'XP

A. Lignes directrices

XP est une méthode basée sur des principes simples, que tout le monde pratique déjà sans le savoir. Mais alors, où est la « révolution » ?

1. Rendre moins lourdes les démarches

Le but d'XP est de trouver des solutions plus ou moins simples, basées sur des principes éprouvés, pour tenter de diminuer les risques pour respecter les délais et la qualité du produit commandé.
Cela, en trouvant un compromis équilibré entre "pas de démarche" et "trop de démarche", tout en respectant le minimum de discipline nécessaire pour attendre un retour sur investissement en temps et en effort immédiat et intéressant.
Ainsi, on souhaite réduire radicalement la production de documents en se focalisant sur un code bien commenté plutôt que sur des documentations techniques très formelles.
XP évite donc avec cela les lourdeurs des méthodes classiques et est donc un habile compromis entre une méthode traditionnelle et le développement « cow boy », sans règles précises. Par conséquent le travail des informaticiens est totalement tourné vers la technique où ils se sentiront vraisemblablement plus à l'aise.

2. Changer les principes

En outre, XP, en tant que méthode agile, se veut adaptative plutôt que prédictive. C'est à dire qu'à l'inverse des méthodes lourdes qui tentent de dresser un plan au début du projet, XP indique qu'il vaut mieux agir sur le court terme. Cela, tout simplement parce qu'il n'existe pas de projet figé où le prédicat de base ou le contexte ne changent pas du tout. Dans ce cas, le coût du changement a une croissance logarithmique en fonction du temps.
Le simple fait de ne pas être réticente aux changements permet dans un premier temps de mieux s'y adapter mais permet aussi une meilleure relation avec le client et la livraison d'un produit conforme totalement aux exigences de ce dernier.
Enfin, le dernier but d'XP est de se vouloir orienté sur les personnes plutôt que sur les processus. Alors, le développement tentera d'être une activité agréable plutôt qu'un cauchemar bureaucratique. Le nombre d'heures supplémentaires sera limité puisque

B. Les 4 valeurs d'XP

Le père d'XP et ses adeptes d'XP définissent quatre valeurs que l'on doit respecter pour entrer dans le clan, pourtant non fermé, des XPiens.

1. Communication

Le manque de communication est dans la vie courant un créateur de malentendu et cela devient encore plus grave dans un projet, pour tous les acteurs. Plusieurs pratiques dans XP ont pour finalité de permettre de se poser les bonnes questions et un partage de l'information entre acteurs du projet : au sein de l'équipe de développement et entre développeurs et clients.

a) Dans l'équipe

La communication avec XP est une valeur essentielle. Le code étant propriété de tous, chacun doit tenir au courant les autres de l'état d'avancement de son travail. Cela permet aussi de demander éventuellement de l'aide ou de répartir les travaux complexes ou plus long. Aussi, grâce à cette communication, chacun partage ses connaissances, ce qui est bénéfique pour l'équipe, le client, l'entreprise, etc ! Ainsi, se crée une réelle et formidable dynamique de groupe et un esprit d'équipe solide et efficace.

Lors des tests d'évaluations, la communication a lieu aussi avec le chef de projet puisque ce dernier s'entretient avec son collègue pour savoir ce qui a été fait et ce qui marche. Par la même occasion, cela permet éventuellement de redéfinir les tâches en fonction de la charge actuelle de travail et cela permet aussi d'avoir une discussion amicale avec tout le monde !
La communication écrite n'est pas en reste puisque le code se doit d'être toujours propre notamment pour permettre une relecture par les autres.

b) Avec le client

En plus de donner des avantages conventionnels au client, XP permet un rapprochement du client par rapport aux informaticiens, notamment lorsque le non-informaticien explique son métier. La communication s'instaure donc plus facilement entre tous les acteurs et les relations humaines n'en sont que meilleures.
C'est d'ailleurs ce qui ressort d'une interview de Dominic Williams, chef de projet à CSEE Transport, qui a du fournir une application à l'entreprise GoldOwner, réalisée par le groupe de réflexion Design-up. Dans celle-ci, M. Williams explique notamment les bénéfices qu'a apporté la méthode dans leur relation avec le client.
En effet, dans la première phase du projet, le client n'a pas été intégré au projet alors que dans une deuxième, il a fait partie intégrante de l'équipe. Selon lui, les rapports et les performances se sont alors beaucoup améliorés, ainsi que le respect des dates butoires qui est devenu au fur et à mesure complètement fiable.
Enfin, le client a beaucoup apprécié la souplesse que la méthode permet pour apporter des modifications aux spécifications en cours de projet (quotidiennement s'il le veut) et la très bonne réactivité pour le traitement des anomalies. La gestion des priorités est aussi plus pertinente puisqu'il peut réorienter les développeurs « en direct ».

2. Feedback

Par le mot feedback, il faut entendre « retours », commentaires, avis…

a) Du client

Tout au long du projet, des retours réguliers du client sont demandés pour progresser. En effet, avec les livraisons régulières, le client donne en continue son avis sur les cohérences du produit avec ses souhaits en terme de fonctionnalité. Le moyen de faire ces feedbacks est la mise en place des tests unitaires qui détectent les bugs et les régressions. En découlent des félicitations, des changements et des améliorations pour coller de plus en plus près à la demande finale. Par conséquent, les développeurs peuvent programmer sans crainte puisque toute divergence sera détecter rapidement.

b) De l'équipe

En parallèle au feedback du client, au sein de l'équipe, les commentaires sont aussi les bien venu. Cela est possible grâce à la programmation en binôme et puisque chacun peut consulter le code de l'autre. Là aussi il en découle des congratulations ou des critiques constructives qui augmentent les compétences du fautif.

3. Simplicité

La vie est déjà assez difficile pour avoir envie de la rendre encore plus qu'elle ne l'est…

a) Le YAGNI

Une autre grande idée d'XP est donc de toujours avoir en tête un principe trivial. Celui-ci porte un nom qui semble complexe mais qui pourtant à des aspirations très élémentaires : le YAGNI (You're NOT Gonna Need It). C'est à dire qu'il faut toujours chercher à supprimer l'inutile pour optimiser au maximum la productivité. Ainsi, un développeur doit toujours avoir en tête cette phrase : "What is the simplest thing that could possibly work?" (« Qu'elle est la solution la plus simple qui peut fonctionner ?").
Cette philosophie rejoint l'idée vue précédemment sur la flexibilité qu'apporte XP aux changements. Cela car, « il vaut mieux faire simple aujourd'hui, quitte à changer demain, plutôt que de faire tout de suite compliqué sans être absolument certain de l'utilité de ce que l'on développe ».

En conséquence, tout au long du développement, la vie du développeur n'est pas de tenter de rechercher une solution exceptionnelle pour les sujets considérés mais son but est plutôt d'évoluer le plus vite possible. Aussi, avec cette souplesse, le client peut demander des modifications sans que le développeur ne soit réticent à l'appliquer, sans non plus d'augmentation considérable des délais.

b) Pour faire des économies

De ce point de vue, XP fait donc le pari que la simplicité d'aujourd'hui revient moins cher et le surcoût éventuel dans le futur est compensé par le fait que, à propos de la solution compliquée, "vous n'en aurez probablement pas besoin". Donc, en moyenne, mieux vaut faire simple.
D'ailleurs cette simplicité est aussi essentielle pour que le code puisse être repris instantanément par n'importe quel développeur pour le compléter ou le changer.

c) Précisions

Attention toutefois, "le plus simple possible" ne signifie pas "simpliste", il ne faut pas se tromper sur ce but ! Par exemple, il ne faut pas se laisser tenter par la simplicité de ne pas reprendre une classe par exemple pour éviter de s'embêter à la relier et la comprendre (ce qui doit d'ailleurs être chose aisée !)… Toute duplication de code est bien sûr interdite !
Enfin, le client se doit aussi d'adhérer à ce principe pour ne pas demander des choses complexes qui ne seront jamais utilisées !

4. Courage

Les acteurs d'un projet qui décident d'utiliser XP, même si la méthode a de grandes qualités, doivent avoir du courage.

a) Avoir confiance en XP

Le client doit avoir le courage de donner des priorités à ses besoins, de dire que tel besoin n'est finalement pas très clair…
Le responsable doit avoir le courage de commencer le projet avant que tout ait été précisé clairement sur un document contractuel. Et même en amont, il doit faire preuve de courage en adoptant une méthode nouvelle, surtout sur des projets stratégiques.

b) Savoir jeter pour repartir sur de bonnes bases

De même pour le développeur qui doit s'obliger à jeter du code qui sent mauvais… Le code a une odeur, quelquefois il faut le changer et tout recommencer. Simplement, XP demande le courage de regarder le développement en face, sans tabou… De plus, la mise en place d'itérations très courtes implique une possibilité de reconsidération régulière de son travail ce qui serait susceptible de décourager certains, voire la plupart !
Enfin, la transparence vis à vis du client et de l'équipe (particulièrement des binômes) demande du courage car chacun peut voir les incompétences de l'autre…

III. PRINCIPES DE MISE EN ŒUVRE

La mise en œuvre d'XP est basée sur des principes que l'on retrouve tout au long de la vie du projet et qui régissent le cycle de vie de ce dernier.

A. Spécifications itératives par le client

Avec des méthodes classiques, plus le temps passe, et plus les réalisations semblent s'éloigner du souhaits (changeant !) du client. Alors le mieux est de réaliser de nombreuses itérations…

1. Diviser pour mieux régner

Cette expression va vraiment dans la philosophie d'XP, si bien sur on ne parle pas le sens péjoratif qui est de diviser l'équipe mais de se séparer pour arriver à de meilleurs résultats !

a) Des spécifications régulières

Tout projet commence obligatoirement par les précisions de la demande du client : les spécifications. Cependant, avec XP, cette première phase ne prend pas le temps généralement constaté. En effet, ne dit-on pas « diviser pour mieux régner » ? C'est tout à fait la façon de penser d'XP.
Plutôt que de tout spécifier au début du projet, le client (on entend par client une personne ayant une vision d'utilisateur final et habilitée à définir les besoins du projet et leurs priorités) va expliquer aux développeurs la première fonctionnalité qui est pour lui la plus importante. Cela se fait lors de réunions nommées les « planning game » où la discussion entre les développeurs et le client est très importante. On y discute des spécifications, des solutions et du planning. Ensuite, les informaticiens s'attachent à implémenter cette première demande dont le temps de développement ne devra dépasser une à deux semaines.

b) De nombreuses itérations

Chaque partie du projet, qui devient très itératif, est ainsi divisée en sous-modules qui doivent être opérationnels au bout d'un intervalle de temps prédéfini relativement court.
A la fin de chaque itération, une mini-livraison est faite permettant au client de déjà « s'amuser » avec le produit, de le tester et de se rendre compte du résultat final. Grâce à ces premiers essais, le client peut demander des améliorations, des réorientations plus ou moins profonde au développeur qui a une grande réactivité puisque tout est encore « chaud » dans sa tête. Une fois que la fonctionnalité est validée, on passe à une autre période de spécification, etc.
Avoir plusieurs changements au fur et à mesure est d'ailleurs une chose préférable à un grand chamboulement !
Dans la pratique, un délai de 3 semaines est généralement choisi par l'équipe pour une fonctionnalité à tester. Ces délais sont bien sûr exhaustifs mais sont généralement considérés comme optimaux par l'équipe et le client.

2. Définition des besoins par des user-stories

Traduit en français, user-stories signifie « histoires de l'utilisateur » et c'est bien ce que ça représente : récit de ce que veut l'utilisateur.

a) Définition

On demande donc au client de définir les spécifications en petites histoires testables en 1 à 2 semaines. Or, n'arrive-t-il jamais que le client de sache pas comment faire ? Après tout, chacun sa spécialité ! En faite, la question ne se pose pas car dans XP, le client utilise des « user stories » pour définir ses besoins. Comme leurs noms l'indiquent, ces fonctionnalités nécessaires à l'utilisateur sont décrites à la façon d'une bande dessinée (contenu simple et imagé), sur des fiches bristol format A5. Elles sont donc courtes et n'abordent pas les considérations techniques pour rendre la production la plus intuitive possible. On doit utiliser des métaphores pour que tout le monde puisse bien se comprendre. Tout doit rester bien concret avec XP !

Inévitablement, les besoins ne sont pas suffisamment précis. Le développeur est donc dans l'obligation de dialoguer avec le client pour obtenir les détails ce qui favorise encore une fois l'établissement d'une réelle communication.

b) Points forts et faiblesses

L'un des avantages de cette technique est que l'utilisateur écrit vraiment ses besoins, alors que dans une approche plus classique le développeur (ou l'ingénieur d'affaire ou le chef de projet) retranscrit plus ou moins bien ce qu'il a compris.
Ainsi, le résultat est fondamentalement différent. En effet, les user stories sont vraiment une technique d'expression des besoins alors que les cas d'utilisation, demandés par les méthodes classiques (UML), se cantonnent souvent à une translation des besoins. De plus, l'implication du client est également sans commune mesure. Dans XP, il est effectivement partie prenante dans le pilotage du projet, itérations après itérations.
Toutefois, le niveau de détail d'une « user story » est inférieur à celui d'un cas d'utilisation UML. Pour cause, elle n'est constituée que d'un titre, d'une ou deux phrases d'explication du fonctionnement voulu et d'un schéma, sans aborder les conditions d'erreurs et les traitements associés.

Aussi, XP se voulant très itératif, son contenu ne pourra excéder un cycle d'itération ou dans le cas contraire, l'user story devra être découpée. A l'opposé, une user story trop légère sera fusionnée avec un autre bristol.

3. Les prévisions détaillées

Le rôle d'un chef de projet est souvent décrit comme la personne qui planifie. Qu'en est-il de cette problématique avec XP ?

a) Des estimations précises

Grâce à ces jalons fréquents, on peut se permettre de prévoir avec une plus ou moins grande précision les délais et la date de fin du projet : chaque fiche correspond à une itération dont la durée est approximativement connue. Cette durée est composée d'unités arbitraires (XP units ou gummy bear) dont la valeur est discutée avec le client et dont le nombre est défini par les développeurs. En effet, seuls ces derniers connaissent précisément le temps qui leur est nécessaire, en fonction de leurs compétences et de leurs expériences, pour aboutir au résultat décrit dans la fiche. D'ailleurs, pour se décider, le développeur peut utiliser son PC pour faire de l'expérimentation, ce qui est nommé « spike solutions ».
De même, on peut prévoir les coûts humains puisque chaque user story engendre un cycle de développement d'une ou deux paires de programmeurs.

b) Des mises à jour régulières

A noter que ces prévisions peuvent, bien sûr, être re-négociées et re-considérées puisque le projet est revu à chaque itération. Cela, pour deux raisons : la première est que tout ne se passe jamais comme prévu (une équipe de développeur peut rester bloqué sur un problème qui ne semblait pas complexe par exemple). La deuxième vient du fait qu'avec XP le client peut se permettre de changer, compléter ou de supprimer sa demande.
Quoi qu'il en soit, plus on arrivera vers la fin du projet et plus les estimations seront précises, notamment avec l'augmentation des compétences des programmeurs leur permettant d'affiner leurs estimations.
Les prévisions reprécisées régulièrement, à chaque itération, permettent de donner aux directions des chiffres précis et en temps réel, pour anticiper des dérapages éventuels et diminuer les risques de dépassement. En connaissance de cause, les responsables pourront ainsi ajouter, si nécessaire, des ressources pour respecter les délais, redéfinir les objectifs ou, même, décider de stopper le projet, s'il semble s'avérer irréalisable, avant que les pertes ne soient trop grandes.

B. Les tests écrits avant le programme

Après avoir obtenu des user stories précises, les développeurs entrent en jeu plus activement. Leur première activité après les spécifications, n'est pas de coder les fonctionnalités mais les tests. « Ecrire les tests puis coder : si vous ne le faites pas, vous n'êtes pas un Extreme Programmer » affirme Kent Beck.
Avec le principe de ne travailler que par itérations courtes, le processus de test se voit pour cela aussi modifié.

1. Ecrire les tests avant tout

XP se veut totalement novateur dans le domaine des tests. En effet, les tests jouent un rôle fondamental dans XP à tel point que leur importance est comparable à celle de la programmation elle-même.
D'ailleurs, on considère que les tests sont une partie intégrante de la documentation puisque que la programmation de conditions de tests est une indication directe sur la façon d'utiliser l'élément testé.

a) Des tests écrits tôt et automatisés

Avec XP, dans la pratique, la mise en place des tests demande une rigueur importante. Le but est de réaliser à chaque itération des tests fonctionnels définis par les souhaits du client. Ainsi, le

développeur écrit donc le programme de test unitaire à partir de souhaits du client et cela, AVANT le code de l'application.

D'écrire les tests avant, permet de ne rien oublier puisque les spécifications viennent d'être exposées par le client. De plus, on code les tests, qui sont garant de la qualité du produit final, quand on n'est pas encore lassé du développement, avant les fatigantes séances de développement…

Une telle rigueur n'est que très rarement demandée dans les méthodes de gestion de projets informatiques. La phase de tests y est souvent complètement disjointe de la programmation et même parfois faite par une toute autre équipe. Or, cela entraîne forcément des pertes de temps puisque le codeur doit prendre le temps de comprendre l'application et parce que le développeur doit se remettre en tête le code à débugger après cette phase de test plus ou moins longue !

Avec XP, les tests sont écrits au début mais sont aussi caractérisés par leur automatisme ! Ainsi, lancer un test ne revient pas à plusieurs tâches laborieuses à faire les unes à la suite des autres. Tout est regroupé dans une fonction « main » qu'il suffit de lancer. Si tout se passe bien, on le sait tout de suite et si un problème survient, nos propres commentaires permettent de savoir d'où vient l'erreur.
En tout cas, il faut noter que le résultat ne peut être que dichotomique : soit le module fonctionne parfaitement bien dans sa globalité et on peut continuer, soit il faut corriger les problèmes avant de passer à autre chose.

b) Les 2 types de tests

XP définit deux types de tests. Les premiers sont les tests de recette ou tests fonctionnels qui sont rédigés par le client, à partir d'un langage formel si possible. Comme leur nom l'indique, ces tests (Acceptance Tests) servent à valider le travail du prestataire, comme on peut le voir dans les projets classiques avec un cahier de recette. Ainsi, il s'agit tout simplement de vérifier que tout fonctionne correctement et est conforme à la demande, avant de donner le dernier chèque… Mais, avec XP, le client définissant au fur et à mesure ses besoins aux développeurs, il n'y a généralement pas de grandes surprises lors de ces tests…

Le deuxième type de test que recense XP est le test unitaire (Unit Testing). On appelle généralement test unitaire, tout test écrit par l'un des programmeurs dans le but de s'assurer du bon fonctionnement de son programme. Avec XP, c'est la même signification sauf qu'ils sont plus nombreux et plus fréquents. Ainsi, au contraire du test de recette, son but n'est pas de satisfaire le client directement mais permet de vérifier la qualité du développement. Ces tests sont, bien sûr, écrits comme les autres avant le codage et sont exécutés après chaque phase de codage.
Ils doivent tester généralement une itération et leur durée d'exécution est comprise entre 1 et 10 minutes. Au-delà, ils doivent être découpés.

Ces tests sont les plus importants car ils garantissent la stabilité du produit et cela, dès le début, pour éviter de partir sur des bases bancales. Pour cause, ils garantissent la qualité du produit mais incitent aussi le développeur à réfléchir sérieusement sur la conception, puisqu'ils sont réalisés avant le code. Ils peuvent être alors considérés comme plus « contractuels » que les tests de recette qui découlent de ceux-là.

Pour écrire ces tests, la démarche est la suivante : on commence par écrire le test puis on l'exécute après la compilation qui garantie la logique du code. Puisque le test est codé avant le programme, le test échouera. Une fois que le code de l'application est écrit, il devra par contre réussir.
Une modification du code (refactoring) ne devra en aucun faire échouer les tests qui garantissent donc, dès le début, la conformité du développement. Par contre, un changement dans le caractère fonctionnel du code, rappellera au développeur que ce changement est peut-être inutile car non demandé par le client.

2. Un produit fiable et toujours opérationnel

Le rêve diront en cœur développeurs et clients… Avec XP, le rêve peut devenir réalité !

a) Des tests conformes à la demande du client

Les tests systématiques assurent que le système en cours de création est constamment opérationnel puisqu'un dysfonctionnement provoqué par une évolution du code source est tout de suite détecté et donc rapidement corrigé.

De ce fait, le client peut toujours connaître l'avancement exact du projet et surtout obtenir des livraisons régulièrement.
Aussi, la livraison, qui peut se faire donc dès la fin du développement, est donc celle d'un produit totalement fonctionnel et sans aucun bug.

En outre, les tests permettent d'analyser les résultats, déterminer ce qui vous ralentit, et améliorer les procédures pour aller plus vite, tout cela, presque en « live ».
Les tests étant écrits avant le codage, une fois que la partie programmation est finie, on peut tester tout de suite le module.
De ce fait, on peut obtenir très rapidement un feed-back du client qui validera la partie ou demandera tout de suite des modifications, tant que tout est « tout chaud » dans la tête du développeur.
On a donc à chaque itération, un raffinage instantané des besoins clients d'autant plus que chacune des modifications demandées est aussi testée jusqu'à une satisfaction totale du client.

Par conséquent, avec ces tests écrits avant le début de la programmation du logiciel, on obtient obligatoirement du code conforme à la demande. On évite aussi les oublie car on ne programme que ce qui va être testé, c'est à dire tout ce qui est demandé.

b) Des tests optimaux pour une meilleure conception

Ecrire les tests d'une classe avant même d'écrire la classe permet aussi de réfléchir à son fonctionnement dans sa globalité et à ses interfaces. Avec ce recul, puisqu'on ne se précipite pas tête baissée dans la technique, on peut remarquer, dès cette phase de conception, les oublis ou manques éventuels des user stories.

D'avoir les tests avant la fin du développement permet aussi de tester régulièrement le code créé, d'un point de vu syntaxique, sémantique mais aussi logique. En effet, pour les langages assembleurs seule la syntaxe était vérifiée, les compilateurs C s'occupe aussi de la sémantique et ceux des langages objets vont encore plus loin : « cet objet est-il bien utilisé ? » , « Cette méthode est-elle bien appelée ? ». Avec les bibliothèques de tests Junit (de java) et CppUnit (de c++), la philosophie des tests performants est encore plus intégrée au langage lui-même !

Mais les tests présents dans les compilateurs ne peuvent aller plus loin et les fonctionnalités ne peuvent être testées. C'est pourquoi d'écrire les tests au début, permet d'avoir systématiquement et automatiquement, après chaque phase de développement, l'assurance d'avoir un module bien conçu et totalement correct.

D'ailleurs, on en arrive à se demander si ce n'est pas grâce aux tests que l'on arrive à programmer mieux et plus rapidement. Pour cause, on possède un garde-fou : si on fait quelque chose de faux, le filet de sécurité des tests intégrés au langage prévient qu'il y a un problème et montre même où il réside…

C. Programmation

Dans une méthode de gestion de projet de développement logiciel, on s'attache souvent à respecter beaucoup de règle d'organisation mais on s'occupe beaucoup moins de la matière première du projet : le code source.

1. Un code propre et efficace

a) Programmer simple…

Pour développer avec XP, il faut respecter des règles précises qui ont beaucoup d'intérêts.
La conception doit être la plus simple possible (*Simple Design)* surtout, au début, puisque le code est successible d'être complètement modifié par la suite, en cas de changement de besoins.
*L*e code créé doit être composé de petites classes et de courtes méthodes. La relecture du code doit être la plus aisée possible. Cela passe aussi par l'utilisation de noms explicites pour les variables, les méthodes, etc. Elémentaire pourrait-on penser, et bien oui ! XP utilise bien des principes de bases « poussés à fond »!
La relecture passe aussi par la présence obligatoire de commentaires dans le code qui sont dans la pratique souvent bâclés. Il faut aussi penser à remettre à jour ces commentaires au même rythme que le code ce qui n'est malheureusement pas toujours le cas. Attention toutefois à ne pas inonder le code de commentaire. On doit insérer dans le code le strict nécessaire.
Aucune duplication de code ne doit exister, conformément au principe DRY (Don't Repeat Yourself) de « The Pragmatic Programmer ».
Grâce à l'application de ces règles, on obtient un code homogène pour toute l'application. Et pourtant, l'équipe est composée d'une dizaine de développeurs de générations différentes !

b) Du code propre grâce au refactoring

XP va plus loin encore dans la clarté du code. En effet, des séances de « refactoring » doivent avoir lieu régulièrement et au moins quotidiennement. Ce terme a été défini et décrit très en détail par Martin Fowler dans son ouvrage « **Refactoring, Improving the Design of Existing Code »** qui n'est pas disponible en version francophone. Sa définition est « toute modification d'un programme permettant d'en améliorer la structure interne sans en modifier le comportement externe ».
Pendant ces périodes, le développeur permet de prendre un peu de recul sur le code et de revoir d'un coté l'architecture du programme et de l'autre, la simplicité du code. Par exemple, il s'agit de décomposer les algorithmes en entités très courtes à placer dans des méthodes distinctes, de donner des noms clairs aux variables et méthodes et surtout de positionner correctement dans l'espace, les blocs formés par les boucles et les conditions.
Ainsi, un « extrême développeur » se doit d'améliorer fréquemment le code existant afin qu'il puisse mieux s'adapter aux exigences du lendemain.

2. L'organisation du développement

Le développement est loin d'être une simple tâche de dactylographie. Cette activité se doit donc d'être bien organisée, et d'autant plus avec XP.

a) Pair Programming

Concernant le développement, une caractéristique d'XP, qui est aussi la plus controversée, est la programmation par binôme, c'est à dire, deux développeurs pour une machine. Et le but n'est pas dans cette démarche de faire des économies de matériel informatique…
Les règles pour réaliser cette pratique sont strictes. Il ne s'agit pas de permettre à l'un de se reposer pendant que l'autre travail. Chacun programme sa partie de code, dont il reste propriétaire. La programmation se fait alors à tour de rôle et l'échange du clavier est régulier, soit à la fin de chaque classe ou quand la fatigue commence à trop peser.
Dans la programmation en binôme, la personne qui ne tape pas au clavier peut réfléchir à la suite, à l'architecture globale de l'application, vérifier la frappe et surtout commenter pour faire évoluer le code. Une personne produit donc le code (le « driver ») tandis que l'autre prend du recul pour vérifier la cohérence globale et réfléchir à la conception générale. Le « partner » garde aussi à porté de main les règles d'XP ! Cela permet aussi d'avoir une autre personne qui sait ce qui a été fait et qui est capable de continuer au cas où (accident ou vacances).
Aussi, il est important que les binômes changent régulièrement, c'est à dire à chaque nouvelle itération d'une à deux jours. Ainsi, chacun travail avec toutes les personnes de l'équipe, ce qui permet forcément de un rapprochement et permet aussi un partage de savoir entre les développeurs. Aussi,

si un binôme reste bloqué, leur discussion peut être entendue par une autre personne de l'open workspace (grand bureau partagé par plusieurs personnes), qui pourra venir apporter son aide.
La première critique qu'on pourrait faire à ce principe est la perte de productivité globale. Pourtant, ce fondement fait la force d'XP car, dans la pratique, le rendement horaire s'approche de 1. Pour cause, la fatigue de l'un est compensée par l'énergie de l'autre et ils s'auto-disciplinent. Aussi, de grandes économies sont réalisées pour la phase de débuggage puisqu'on s'offre une relecture constante du code.

b) Des développeurs heureux

XP se veut une méthode humaine et plutôt orientée vers le conforme des développeur et cela pou plusieurs raison. La première est que la communication est grande et notamment grâce aux « stand-up meeting », réunions debout (pour s'assurer qu'elles ne s'éternisent pas) quotidiennes qui permettent d'exposer la situation de chacun, les éventuelles demandes d'aide… Aussi, le développeur a la possibilité de choisir l'itération qu'il souhaite traiter, à condition bien sûr que le sérieux de l'équipe soit assez grand pour ne pas laisser traîner une tâche lassante qui peut ralentir l'avancée globale du projet. Ainsi, les développeurs ne se sentent pas frustrés par l'autoritarisme d'un boss, qui ne fait alors plus que motiver son équipe…
Chacun ne peut aussi refuser d'apporter son aide à un collègue !

Grâce à tout cela, naît aussi un réel esprit d'équipe et réelle dynamique de groupe. Les développeur joue pour gagner (et non pour participer) et ce la fonctionne !
Dans la pratique, on a pu aussi constater que les développeurs étaient plus motivés, heureux et compétents.
Enfin, XP préconise de ne pas réaliser d'heures supplémentaires (si si !), en tout cas, jamais deux semaines de suite. Une semaine de travail ne devra qu'exceptionnellement dépasser 40 heures (comme pour les "40-hour week" des Etats-Unis). En effet, une telle surcharge indique un problème structurel, de planning, plutôt qu'un problème de productivité. De plus, la réalisation d' « heures sup » ne peut qu'avoir un effet néfaste sur la productivité, la motivation et surtout sur la qualité du code !

D. Documentation

Avec les commentaires du code pourtant complets, il faut quand même prévoir une petite documentation. En effet, XP n'est pas anti-documentation, par contre, la méthode se veut très vigilante la quantité des documents.

1. Uniquement deux types de documents

Par conséquent, il faut bien se demander quels documents sont réellement utiles. A ce sujet, il faut distinguer deux types de documents correspondant aux deux types de destinataires, les informaticiens et le client. Pour ces derniers, ils ne peuvent être réduits voir supprimer puisqu'ils apparaîtront dans le décompte final de la facture.
Par contre, une importante réflexion doit avoir lieu avant de commencer toute rédaction. En effet, il faut bien avoir en tête que de bons commentaires devraient suffisant et surtout que le travail en équipe et plus particulièrement par deux tels que le définit XP, ne devraient pas engendrer de documentation complémentaire.

Aussi, la création de documentation ne permet en aucun cas d'augmenter la vitesse de programmation, de rendre le système plus fiable ou même de le rendre plus facile à maintenir. Les seuls intérêts qui pourraient être envisagé seraient une éventuelle aide à la relecture par une autre équipe, une éventuelle lecture pour les clients ou enfin un support pour la publication d'articles. Or, ces cas ne semblent que guère fréquents, voir exceptionnels.

2. Des formes bien précises…

Toutefois, il peut arriver qu'une documentation soit nécessaire dans un cas où la globalité du projet serait difficile à voir. Mais dans ce cas, il faut bien s'interroger sur l'utilité effective de ce document qui

ne doit être en aucun cas être redondant par rapport à un éventuel dossier existant. Aussi, le document final ne devra en aucun cas excéder dix pages et devra mixer textuel, diagrammes (UML par exemple mais épurés) et surtout rediriger vers les fichiers sources (le code) et leurs « javadoc » (documentation au format HTML créé automatiquement par java, à partir des commentaires du code). Notons bien qu'on pourra s'aider des tests qui mettent en avant les fonctionnalités essentielles de l'application.
Pour conclure, concernant les spécifications pour la documentation, on retrouve bien le principe YAGNI de XP avec la suppression de tout travail inutile. Encore une fois, ce principe tout simple est une grande innovation puisque dans les autres méthodes, une documentation de plusieurs dizaines de pages est demandée alors qu'elle ne sera probablement jamais lue par personne.

E. Intégration - Livraison

Le fait qu'XP ne permettent pas réellement de prévoir à long terme les échéances, ne pose en aucun cas un problème pour les livraisons.
En effet, les intégrations des classes développées sont faites plusieurs fois par jours ou, au moins tous les soirs ! Les tests sont alors lancés pour vérifier la cohérence et la non-régression du produit.
Cela à l'avantage d'avoir un produit fonctionnel toujours à jour, ce qui est permit par le fait que les tests sont écrits auparavant. Ainsi, un « eXtreme Programmer » ne sera que très rarement contraint de passer des week-ends ou des nuits à développer pour respecter une deadline. Cela permet aussi de permettre à l'équipe et au client de connaître précisément l'avancement.
A tout moment, on peut donc fournir au client un produit utilisable, bien que les livraisons soient prévues environ tous les 3 mois. Chaque trimestre, un module entier et complètement fonctionnel est donc réceptionné et validé par le client, ce qui permet d'obtenir des paiements (restons concret tout de même !).

IV. LES LIMITES D'XP

Extreme Programming apparaît comme la plus radicale des méthodes agiles. De ce fait, elle draine autour d'elle d'"extrémistes" et farouches opposants. Parmi eux, le plus actif est sans conteste Matt Stephens qui critique la méthode dès qu'une occasion se présente (notamment sur plus d'une trentaine de pages sur internet d'après Google).

A. Des équipes particulières

1. Une composition d'experts

Les buts d'XP sont de palier à des carences constatées dans les projets informatiques. Cependant, il ne peut y avoir d'évolutions sans compétences de ses acteurs.
C'est pourquoi, les premières limites à l'utilisation de cette méthode sont les capacités techniques de chaque membre de l'équipe et leurs capacités relationnelles. En effet, chacun devra être capable d'assumer une productivité optimale (pendant 35 heures par semaine, soit…) puisque règne le « death march », c'est à dire une grande conscience professionnelle pour finir dans les délais le projet.
Aussi, le niveau technique exigé pour un développement avec XP ne demande pas qu'une bonne connaissance dans le langage de programmation mais aussi en génie logiciel. En effet, pour construire des choses solides, il faut avoir mis en place une architecture habile et puissante.
De plus, du fait de la perpétuelle remise en question possible par le client du produit, clients et développeurs sont disposés à faire évoluer leurs façons de faire. Cela demande encore une équipe courageuse et très ouverte.
Enfin, avec cette méthode moins structurée, certains développeur, surtout les plus jeunes. L'absence d'un cadre formel structurant n'apporte pas l'aide que peut fournir une méthodologie offrant plus de repères
De même, la plupart des responsables peuvent ne pas se sentir à l'aise ou redouter plus les pépins…
Finalement, les membres de l'équipe doivent donc être performants et polyvalents, ce qui n'est pas forcément innés chez tous les ingénieurs.

2. Une taille d'équipe limitée

Jusqu'à présent, XP n'a été utilisé que par des équipes de 2 à 10 développeurs.
Au-delà de 12 personnes, certaines pratiques de XP ne peuvent plus être adaptées en raison de l'alourdissement des procédures et des problèmes de communication qui apparaissent. Cela est contraire aux principes même d'XP !
Alors, aussi efficace que la méthode puisse être, elle ne peut malheureusement pas se généraliser à toutes les organisations et donc à toutes les entreprises.

3. Des experts en relationnels

Avec XP, chacun devra faire preuve d'un bon esprit relationnel pour discuter et négocier avec le représentant du client, c'est à dire la personne qui paye !
Aussi, la communication doit être bonne avec l'équipe et plus particulièrement le binôme, qui change en permanence, ce qui oblige à travailler avec des personnes de tout âge, ou qu'on n'apprécie pas forcément…
Enfin, il ne faut pas négliger le fait que certains développeurs sont littéralement "paralysés" dès lors qu'ils se sentent observés ou seulement moins sérieux…

B. Pour des projets particuliers

1. Le code, source de conflit ?

Avec XP, une classe appartient à une seule et unique personne, ce qui la responsabilise mais le code de l'application appartient à toute l'équipe et est donc consultable par tous. Cela a l'avantage de permettre à tous, une connaissance et une visibilité globale du projet. Chacun est soucieux de la qualité globale du projet et peut être fier de sa contribution.
Cependant, si le niveau des développeurs est peu homogène, cela peut entraîner, du fait de la confiance qu'accorde XP aux développeurs, des pertes de qualité. Aussi, des critiques peuvent apparaître, voire de conflits, en raison du refactoring qui permet à chacun, dans un but d'optimisation, de modifier le code, ce qui peut en vexer plus d'un…
XP, par ses exigences, implique aussi une restriction dans le choix des langages à utiliser. En effet, le choix devra se porter sur un langage objet qui favorise des itérations courtes et qui permet la mise en œuvre facile de tests.
Parmi ceux correspondant à cette caractéristique, on trouve le Smalltalk, Python ou Perl, et plus généralement les langages de scripts mais qui sont de moins en moins enseignés.
Aujourd'hui, ceux qui ont « remportés » le choix des XPiens sont donc le C++ et le JAVA. Toutefois, le premier est reproché pour sa difficulté de débuggage (allocation mémoire, erreurs de compilation, d'exécution…) et commencent d'après certains à faire un peu vieillot par rapport au miraculeux java. (Pour plus d'info sur les différences en java et le C++ : xpose.free.fr) Avec ces clases JUnit, c'est donc JAVA qui apparaît comme le plus adapté. D'ailleurs, XP est décrit et conseillé dans le livre « Thinking in Java ».
Mais dans la pratique, tous les développeurs partagent-ils tous cette préférence ? La culture d'entreprise a-t-elle déjà intégré l'utilisation de JAVA ?

2. Pour des projets de petite envergure

XP ne peut s'adresser qu'à des projets petits ou moyens en raison de ses caractéristiques.
En effet, une itération de trois semaines ne semble pas réaliste pour une équipe de 100 développeurs, trois semaines étant approximativement le délai pour, tout simplement, contacter et informer l'ensemble des personnes…
La communication, le feedback, les itérations courtes ne peuvent s'accommoder à un grand projet.
Aussi, l'investissement de tous est demandé pour un projet XP. Quand l'équipe s'agrandit cette demande n'est-elle pas trop utopique ?
A nouveau, les principes même de XP ne permettent donc pas de généraliser la méthode à l'ensemble du monde informatique.

3. Le client sur place

XP demande à l'organisation de pratiquer des tests très peu espacés dans le temps et en présence d'un représentant du client afin d'avoir un feedback constant. D'autant plus en raison du mode de données des spécifications et de l'intégration quotidienne qui obligerait le client à conserver un de ses responsables pendant des mois avec les développeurs, sans qu'il soit réellement productif.
Ainsi, et c'est probablement un des freins à l'adoption de XP, une mutation dans le rapport client-développeurs doit avoir lieu pour respecter les principes de la méthode puisqu'elle suppose une souscription du client à la plupart des journées de développement.
Le client doit donc être intégré à l'équipe de développement, être dans le même bureau (*On-Site Customer)*, ce qui se marie mal avec le principe même de la sous-traitance qui est une grosse partie du marché de l'informatique…