Table des matières

Aller plus loin

L'eXtremme Programming (JL Bénard, L. Bossavit, R.Médina, D. Williams) - Ed. EYROLLES

www.extremeprogramming.org
www.xprogramming.com
http://xp-france.net/cgi-bin/wiki.pl?ExtremeProgramming
www.design-up.com
www.idealx.org/doc/xp-synthese.fr.html

Introduction

L'**eXtremme Programming** (**XP**) est un ensemble de pratiques qui couvrent une grande partie des activités de la réalisation d'un logiciel : planification, organisation de l'équipe de développement, échanges avec le client, programmation.

Ces pratiques d'XP n'ont rien de révolutionnaire : il s'agit simplement de pratiques de bon sens mises en oeuvre par des développeurs ou des chefs de projet expérimentés.

La nouveauté introduite par XP consiste à pousser ces pratiques à **l'extrême** (d'où le nom de la méthode) et à les organiser en un tout cohérent.

En résumé, l'eXtreme Programming (XP) est une méthode de développement légère conçue pour de petites équipes confrontées à des environnements mal connus ou fortement changeants.

Cette méthode mise au point sur le terrain propose un processus focalisé sur les objectifs élémentaires du développement :

Développer vite - Développer juste.

La méthode de développement Extreme Programming

Les phases

Il y a trois moments (ou phases) dans un cycle de développement de Extreme Programming. Chaque cycle d'une durée ne dépassant pas deux semaines produit une *release* fonctionnelle (une livraison) du produit :

- L'**exploration** : définition des besoins du moment.
- L'**engagement** : choix des besoins qui vont être implémentés en fonction de la valeur que leur accorde le client.
- Le **pilotage** : implémentation des besoins et feedback immédiat des conditions du développement, ainsi que l'impact du produit afin d'orienter la phase d'exploration du cycle suivant.

Les variables

XP considère que la satisfaction de quatre variables va contribuer au développement efficace d'un produit fonctionnel. Ces variables sont :

- Le **coût** : le développement se fait par petits modules. Il est donc aisé de suivre de près les dépenses et ainsi d'agir en conséquence pour ne pas dépasser les budgets.
- Le **temps** : de la même manière que pour le coût, le développement par modules permet de suivre l'évolution du développement en fonction du temps passé.
- La **qualité** : pour XP, la qualité doit toujours être maximum. C'est la seule variable qui ne peut pas varier.
- L'**étendue** : le nombre de fonctionnalité va dépendre du temps et du budget à disposition. C'est le client qui va décider de ce qui sera développé ou pas afin de garder la plus grande qualité possible, en restant dans les délais et les budgets.

Les valeurs

Afin de satisfaire les quatre variables, Extreme Programming met en évidence quatre valeurs essentielle au travers de la méthode : la **communication**, la **simplicité**, le ***feedback*** et le **courage**.

Les activités

Les activités de XP sont les étapes minimales garantissant la réussite du développement du projet : Il faut **coder** et **intégrer**, car à la fin de chaque itération, il doit y avoir une nouvelle *release* fonctionnelle pour le client. Il faut **tester** pour rendre la *release* réellement fonctionnelle. Il faut **écouter** les besoins du client, ainsi que les changements qu'il veut apporter. Il faut **modéliser** (faire du *design*) afin de réaliser l'architecture la plus simple possible.

Les pratiques d'XP

XP définit **treize pratiques**, classés ici en trois catégories : celles relatives à la programmation, celles relatives au fonctionnement interne de l'équipe et celles qui sont liées à la planification et aux relations avec le client.

Les pratiques de programmation

Conception simple (*simple design*) : les développeurs implémentent toujours la solution la plus simple qui puisse fonctionner. En particulier, ils n'inventent pas de mécanismes si le besoin immédiat ne l'exige pas.
Important : conserver une conception simple est un travail difficile et demande beaucoup de rigueur.

Remaniement (*refactoring*) : les développeurs n'hésitent pas à revenir sur le code écrit pour le rendre plus «propre», le débarrasser d'éventuelles parties inutilisées, et le préparer à l'ajout de la fonctionnalité suivante. D'une manière plus générale, cette pratique propose une démarche de conception continue qui fait émerger la structure de l'application au fur et à mesure du développement.

Développement piloté par les tests unitaires (*test-first programming, unit tests, developer tests*) : les développeurs écrivent des tests automatiques pour le code qu'ils produisent, et ce au moment même d'écrire le code en question. Cela leur permet d'une part de mieux cerner le problème avant d'écrire le code, et d'autre part de constituer progressivement une batterie de tests qui les autorise ensuite à apporter rapidement des changements dans l'application, tout en conservant une certaine sérénité.

Tests de recette (*acceptance tests, customer tests*) : le client précise très explicitement ses besoins et les objectifs des programmeurs en participant à la rédaction de tests de recette. Comme les tests unitaires, les tests de recette doivent être automatiques afin de pouvoir vérifier tous les jours la non-régression du produit.

Les pratiques de collaboration

Programmation en binôme (*pair programming*) : lorsqu'ils écrivent le code de l'application, les développeurs travaillent systématiquement à deux sur la même machine (il s'agit là d'une forme «extrême» de relecture de code), dans laquelle les deux développeurs collaborent activement pour résoudre les problèmes qu'ils rencontrent. Les binômes changent fréquemment, ainsi chacun est amené à travailler tôt ou tard avec tous les autres membres de l'équipe.

Responsabilité collective du code (*collective code ownership*) : tous les développeurs de l'équipe peuvent être amenés à travailler sur toutes les parties de l'application. De plus, ils ont le devoir d'améliorer le code sur lequel ils interviennent, même s'ils n'en sont pas les auteurs initiaux.

Règles de codage (*coding standards*) : les développeurs se plient à des règles de codage définies par l'équipe elle-même, de manière à garantir l'homogénéité de leur code avec le reste de l'application, et ainsi à faciliter l'intervention d'autres développeurs.

Métaphore (*metaphor*) : les développeurs n'hésitent pas à recourir aux métaphores pour décrire la structure interne du logiciel ou ses enjeux fonctionnels, de façon à faciliter la communication et à assurer une certaine homogénéité de style dans l'ensemble de la conception.
Nommer de manière cohérente les classes, les méthodes, les données, les fonctions ... Cela facilite la compréhension et la réutilisabilité du code.
Aucune connaissance technique spécifique ne doit être nécessaire à la compréhension du code.

Intégration continue (*continuous integration*) : les développeurs synchronisent leurs développements aussi souvent que possible, au moins une fois par jour. Cela réduit la fréquence et la gravité des problèmes d'intégration, et permet de disposer à tout moment d'une version du logiciel qui intègre tous les développements en cours.

Les pratiques de gestion de projet

Livraisons fréquentes (*frequent releases*) : l'équipe livre des versions du logiciel à un rythme régulier, aussi élevé que possible, la fréquence précise étant fixée par le client. Cela permet à l'équipe comme au client de s'assurer que le produit correspond bien aux attentes de ce dernier et que le projet est sur la bonne voie.

Planification itérative (*planning game*) : la planification du projet est réalisée conjointement par le client et l'équipe de développement, au cours de séances dédiées, organisées régulièrement tout au long du projet.

Client sur site (*on-site customer, whole team*) : le client est littéralement intégré à l'équipe de développement pour arbitrer les priorités, et définir précisément ses besoins, notamment en répondant en direct aux questions des programmeurs et en bénéficiant du *feedback* immédiat d'une application aux livraisons fréquentes.

Rythme durable (*sustainable pace*) : l'équipe adopte des horaires qui lui permettent de conserver tout au long du projet l'énergie nécessaire pour produire un travail de qualité et mettre en oeuvre efficacement les autres pratiques.

Le cycle standard XP

1. Le client écrit ses besoins sous forme de scénarios.
2. Les développeurs évaluent le coût de chaque scénario, en collaboration avec le client.
3. Le client choisit les scénarios à intégrer à la prochaine livraison.
4. Chaque développeur prend la responsabilité d'une tâche pour la réalisation d'un scénario.
5. Le développeur choisit un partenaire.
6. Le binôme écrit les tests unitaires correspondant au scénario à implémenter.
7. Le binôme prépare l'implémentation en réorganisant le code existant, puis il procède à l'implémentation proprement dite.
8. Le binôme intègre ses développements à la version d'intégration.

Planifications collectives (*planning game*)

Le projet est décomposée en une suite de « tours » appelés itérations (durée de 1 à 3 semaines selon les projets) et de livraisons au rythme d'une livraison toutes les 2 ou 3 itérations en général.

Cycle des livraisons

Le cycle des livraisons porte sur la planification des fonctionnalités décrites sous forme de scénarios client (*user stories*).

Une livraison se décompose en trois phases :

- phase d'exploration : définition et estimation des scénarios client
- phase d'engagement : répartition des scénarios pour établir le plan des livraisons
- phase de pilotage : réalisation des scénarios en suivant le cycle des itérations

Cycle des itérations

Le cycle des itérations porte sur la planification des activités (tâches) menées par les développeurs.

Une itération se décompose en trois phases :

- phase d'exploration : définition des tâches (décomposition des scénarios client)
- phase d'engagement : répartition des tâches entre les développeurs
- phase de pilotage : réalisation des tâches

Tâches

Les tâches sont donc les activités des développeurs qui permettent de réaliser un scénario client. Elles sont définies par les développeurs selon des considérations techniques. Un binôme peut réaliser une tâche en une ou deux journées.