XP eXtreme programming

Extreme Programming (XP) est une méthode de développement agile orientée sur l’aspect réalisation d’une application. Elle propose un cadre pour l'ensemble des aspects du projet logiciel, depuis l'analyse des besoins jusqu'aux tests, en passant par la conception.

Histoire :

Elle a été conçue à l’origine par Kent Beck, Ward Cunningham, Ron Jeffries et Palleja Xavier pendant leur travail sur un projet de calcul des rémunérations.   
Elle est née officiellement en 1999 avec le livre Extreme Programming Explained de Kent Beck.

Elle est définie comme :

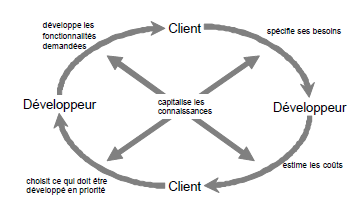
- une tentative de réconcilier l'humain avec la productivité  
- un mécanisme pour faciliter le changement social   
- une voie d'amélioration   
- un style de développement   
- une discipline de développement d'applications informatiques

Son but principal :

Réduire les coûts du changement.

# Valeurs :

**- Communication**  
Pour éviter le plus de problème. Les tests, la programmation en binôme et le jeu du planning (qui sont des principes de base), obligent l’équipe à communiquer très fréquemment.



Interactions entre les acteurs d’un projet XP extreme programming

**- Simplicité**  
Une application simple est plus facile à faire évoluer.

**- Feedback** (retour d’informations)  
Il est primordial pour le programmeur et le client. Les livraisons fréquentes et les tests unitaires et fonctionnels permettent un feedback fréquent.

**- Courage**  
Le client doit avoir le courage de donner un ordre de priorité à ses exigences et de reconnaître que certains de ses besoins ne sont pas toujours très clairs.  
Le développeur doit avoir le courage de modifier l’architecture même si le développement est déjà bien avancé ou de jeter du code existant et d’accepter qu’il est parfois plus rapide et efficace de réécrire une portion de code à partir de zéro plutôt que de modifier celui existant.

Le courage permet de sortir d'une situation inadaptée. C'est difficile, mais la simplicité, le feedback et la communication rendent ces tâches accessibles.

**- Respect**Les programmeurs ne devraient jamais valider les modifications qui cassent la compilation ou qui font échouer les tests unitaires existants ou qui retardent le travail de leurs pairs.   
Les membres doivent respecter leur propre travail en recherchant toujours la qualité et la meilleure conception pour la solution.

# Principes de la programmation :

Les principes utilisés dans cette méthode ne sont pas nouveaux, mais la particularité de cette méthode est de **les pousser à l’extrême.**

**- Planning game :**  
Le client crée des scénarios pour les fonctionnalités qu'il souhaite obtenir. L'équipe évalue le tempsnécessaire pour les mettre en œuvre. Le client sélectionne ensuite les scénarios en fonction des priorités et du temps disponible.

**- Petites releases :**D’une version à l’autre, l’évolution doit être la plus petite possible tout en apportant le plus de valeur ajoutée.

**- Utilisation de métaphores :**  
On utilise des métaphores et des analogies pour décrire le système et son fonctionnement. De telles images permettent à tout le monde (même ceux sans grandes compétences techniques) d’avoir une vision globale du système et d’en comprendre les éléments principaux ainsi que leurs interactions.

**- Conception simple :**Il faut mettre en œuvre les scénarios sélectionnés par le client et uniquement cela. Envisager les prochaines évolutions ferait perdre du temps. Plus l'application est simple, plus il sera facile de la faire évoluer à l’avenir. (Ceux qui pratiquent extreme programming résument cela sous la phrase YAGNI (« You ain’t gonna need it », « Tu n’en auras pas besoin »)).

**- Tests (unitaires et fonctionnels)** :  
Avant de mettre en œuvre une fonctionnalité, le développeur écrit un test (unitaire) qui vérifiera que son programme se comporte comme prévu. À chaque modification du code, on lance tous les tests écrits par tous les développeurs, et on sait immédiatement si quelque chose ne fonctionne plus.

Les tests fonctionnels sont créés à partir des scénarios définit par le client. Ils permettent de vérifier le fonctionnement global du système, de contrôler l’évolution du projet et d’affiner l’expression des besoins.

**- Refactoring du code (remaniement du code) :**Amélioration régulière de la qualité du code sans en modifier le comportement. On retravaille le code pour repartir sur de meilleures bases tout en gardant les mêmes fonctionnalités. Les phases de refactoring n'apportent rien au client mais permettent aux développeurs d'avancer dans de meilleures conditions et donc plus vite.

**- Programmation en binôme :**La programmation se fait par deux. Le premier appelé driver (ou pilote) tient le clavier. C'est lui qui va travailler sur la portion de code à écrire. Le second appelé partner (ou copilote) est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes.   
Les développeurs changent fréquemment de partenaire ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

**- Appropriation collective du code :**L'équipe est collectivement responsable de l'application. Chaque développeur peut faire des modifications dans toutes les portions du code, même celles qu'il n'a pas écrites. Les tests diront si quelque chose ne fonctionne plus.

**- Intégration continue :**  
Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.

**- Pas de surcharge de travail :**L'équipe ne fait pas d'heures supplémentaires. Si le cas se présente, il faut revoir le planning. Un développeur fatigué travaille mal.

**- Client sur site :**  
Le client ou un représentant doit être présent pendant toute la durée du projet. Il doit être disponible afin de pouvoir orienter l’équipe sur le résultat souhaité.

**- Standards de code :**Toute l’équipe va travailler sur la totalité du code. Il faut donc établir et respecter des normes de nommage pour les variables, objets, classes, etc.

# Conclusion :

C’est une méthode de management de projet adapté aux équipes réduites avec des besoins changeants.

- Une planification très souple à court terme (deux semaines max) qui permet une estimation des coûts plus simple et plus précise.

- Livraison rapide des prototypes très tôt et à une fréquence élevée permettant l’évaluation des fonctionnalités réalisées et pour maximiser l'impact des retours utilisateurs

- L'équipe de développement travaille en collaboration totale sur la base de binômes

- Le code est testé et nettoyé tout au long du processus de développement

- Des indicateurs permettent de mesure l'avancement du projet afin de permettre la mise à jour du plan de développement