XP

- Principes d'XP :
  - un représentant du **client est intégré à plein temps** dans l'équipe de développement afin d'atteindre une réactivité optimale, aussi bien pour la définition du besoin que pour la validation des livraisons,
  - la programmation se fait en **binôme** ce qui permet l'appropriation de l'ensemble du code par l'équipe de développement et le partage de l'expertise,
  - l'écriture des **tests unitaires se fait avant l'étape de codage**, ce qui améliore leur efficacité et donc la qualité du code,
  - **l'intégration est continue** dès le début du projet, ce qui supprime les risques associés à cette discipline,
  - des **livraisons fréquentes** permettent d'accélérer la convergence du produit par rapport aux besoins des utilisateurs.

XP

- *Extreme Programming* :
  - Les revues de codes sont une bonne chose ➔ en permanence (programmation en binôme)
  - Les tests sont une bonne chose ➔ tout le monde en fera (test unitaire) y compris le client (recette)
  - La conception est une bonne chose ➔ activité quotidienne (refactoring)
  - La simplicité est une bonne chose ➔ vérifiera que c'est le cas
  - L'architecture est importante ➔ tout le monde y contribuera
  - Les tests d'intégration sont une bonne chose ➔ plusieurs fois par jour
  - Les itérations courtes sont une bonne chose ➔ itérations très très courtes

Qu’est-ce que XP ?

- *Extreme Programming* (XP) est une méthodologie destinée à des équipes de **petite et moyenne** taille, qui développent des logiciels dans un contexte où les **besoins sont vagues ou changent rapidement**
- C’est aussi une façon de développer des logiciels, une façon dégraissée, efficace, **peu exposée au risque**, flexible, prévisible, scientifique et **amusante**

Qu’est-ce que XP ?

- Ce que les autres méthodes ne sont pas ou n’ont pas :
  - **Feedback** concret
  - Incrémentale
  - Planification de l’implémentation de fonctionnalités de façon **flexible** et réactive
  - Tests **automatisés**
  - Communication orale, par les **tests et le code**
  - Démarche de conception évolutive
  - Collaboration étroite
  - Pratiques **compatibles** avec les intérêts à court terme des programmeurs et à long terme du projet

Le problème fondamental : les risques

- Dépassement de délais ➔ cycle court
- Abandon de projet ➔ petite livraison
- Détérioration du système ➔ série exhaustive de tests
- Taux de défaillances ➔ tests routine par routine et client
- Incompréhension de l'aspect fonctionnel ➔ le client est là
- Changement des besoins fonctionnels ➔ raccourcir le cycle de livraison
- Fausse fonctionnalité ➔ les tâches à haute priorité seront réalisées
- Turnover ➔ encourage le contact humain + trace…

Analyse économique du développement

- Trois éléments :
  - Flux de trésorerie entrants et sortants
  - Taux d'intérêt
  - Durée de vie des projets
- Le projet est vu comme un ensemble d'option :
  - **Annuler** ➔ extraire tout de même de la valeur
  - **Changer** ➔ autoriser le changement de direction pendant le projet
  - **Différer** ➔ attendre une meilleure situation pour avancer
  - **S'agrandir** ➔ croître rapidement pour bénéficier d'un marché

Les quatre variables

- **Coûts** ➔ beaucoup d'argent pour booster le projet mais trop ne résout rien
- **Délais** ➔ prendre plus de temps peut améliorer la qualité et augmenter le périmètre mais le **mieux est l'ennemi du bien**
- **Qualité** ➔ très difficile à contrôler, attention aux coûts !
- **Périmètre** ➔ le réduire augmente généralement la qualité mais que dit le client ? permet aussi de livrer tôt
- L'annulation de fonctionnalités importantes fait gagner du temps mais le client est perturbé donc XP à 2 stratégies :
  - Acquérir beaucoup d'expérience pratique pour que les **prévisions soient le plus réalistes possible** (planning)
  - Implémenter les fonctionnalités **spécifiées importantes** par le client

Les quatre valeurs

- **Communication** ➔ dire et écrire les choses importantes et les autres
- **Simplicité** ➔ quelle est la solution simple qui marche, on la fait
- **Feedback** ➔ avoir un *feedback* concret sur l'état actuel est système est une priorité absolue
- **Courage** ➔ déchaînez-vous et mettez le turbo

Les principes fondamentaux

- **Accélérer le feedback** ➔ apprendre, apprendre, apprendre, l'information est une composante critique de l'apprentissage
- **Supposer la simplicité** ➔ tout problème est simple ou presque
- **Changer de façon incrémentale** ➔ on ne peut pas tout changer en une fois : risque trop grand
- **Accueillir le changement** ➔ il ne faut pas le subir

Les principes fondamentaux ou presque

- **Chercher la qualité ➔** on doit obtenir soit excellent, soit trop excellent
- **Capitaliser toute expérience ➔** bonne ou mauvaise, tout est bon
- **Réduire l'investissement initial ➔** trop au début ne sert à rien
- **Jouer pour gagner ➔** …
- **Expérimenter dans le concret ➔** mettre à l'épreuve les idées
- **Communication ouverte et honnête ➔** plus rapide et plus serein
- **Exploiter et non rejeter l'instinct des gens ➔** des commentaires ?
- **Accepter ses responsabilités ➔** mettre son nom
- **Adapter les principes aux circonstances ➔** tout bouge tout le temps
- **Mesurer sans tricher ➔** cela se saura tôt ou tard

Retour aux sources

- **Coder** ➔ il faut bien produire le logiciel !
- **Tester** ➔ il faut qu'il fonctionne
- **Écouter** ➔ il faut accepter la critique, on ne décide pas
  « Les programmeurs ne savent rien. Ou plutôt, les programmeurs ne savent rien qui soit considéré comme intéressant par les décideurs. D'ailleurs, si ces décideurs pouvaient se passer des programmeurs, ils nous mettraient à la porte sans perdre une seconde ! »
- **Concevoir** ➔ créer des structures qui organisent de façon logique les parties du système

« On code, donc, parce que tant qu'on n'a pas codé, on n'a strictement rien fait. On teste parce que si on teste pas, on ne sait pas quand on a fini de coder. On écoute, parce que si on n'écoute pas, on ne sait ni ce qu'on doit coder ni ce qu'on doit tester. Et on fait de la conception pour pouvoir continuer indéfiniment à coder, à tester et à écouter. Voilà la liste complète des activités que nous devons aider à structurer »

XP : la solution

- Le **point de départ**
  - La parabole de la leçon de conduite
  - Les **quatre valeurs** : communication, simplicité, *feedback*, courage
  - Les **principes** : accélérer le *feedback*, supporter la simplicité, changer de façon incrémentale, accepter le changement
  - Les **quatre activités fondamentales** : coder, tester, écouter, concevoir

Les pratiques

- Le jeu du planning : Déterminez **rapidement le périmètre** de la livraison (avec priorités). Si la **réalité est loin du planning**, refaites-le
- Petites livraisons : Mettez **rapidement en production**
- Conception simple : la **complexité est éliminée** dès qu'on la voit
- Tests : tests unitaires en **permanence**
- Refactoring : restructuration du système **dès que nécessaire**
- Programmation en binômes : sur un **même poste**
- Responsabilité collective du code : tout le monde **touche à tout**
- Intégration continue : sinon c'est **impossible** à faire
- Heures de travail : **35 heures** par semaine pas plus, pas 2 semaines
- Client sur site : un **utilisateur**, un vrai !
- Règle de codage : faciliter la **communication interne**

Les pratiques

- Intervenant métier : le jeu du planning
  - Quelle est **l'importance** du problème à résoudre (le client est roi)
  - **Classer** les fonctionnalités par priorités décroissantes (idem)
  - **Décomposition** en livrables : le développeur fait faux
  - **Dates** de livraison : dates importantes pour livrer ou à éviter
- Intervenant techniques
  - **Estimation** du temps pour une fonctionnalité donnée
  - **Conséquences** : les choix fonctionnels sont parfois très lourds
  - **Démarche** : organisation du travail et de l'équipe
  - **Planning détaillé** : dans une livraison, que faut-il faire, quand et comment

Organisation

- L’espace de travail est important
  - Il faut de la **place pour les binômes**
  - Un lieu **d’intégration accessible et visible** de tous
  - Un **lieu pour le client**
  - **Personne ne doit gêner personne et tout le monde est à la disposition de tout le monde**

Fonctionnel vs Technique

- *« Le projet doit être guidé par des décisions métier, mais qui doivent se prendre sur la base des décisions techniques en matières de coûts et de risques »*
- **Les intervenants fonctionnels doivent prendre les décisions pour lesquelles ils ont été formés**
  - **Périmètre ou délais** de chaque livraison
  - **Priorité relatives** des fonctionnalités proposées
  - **Définition exacte** des fonctionnalités proposées
- Les intervenants techniques doivent prendre les décisions pour lesquelles ils ont été formés…

Fonctionnel vs Technique

- Les intervenants fonctionnels doivent prendre les décisions pour lesquelles ils ont été formés…
- **Les intervenants techniques doivent prendre les décisions pour lesquelles ils ont été formés**
  - **Estimations du temps** nécessaire pour l'implémentation
  - **Estimations des conséquences** de divers choix techniques
  - Démarche de développement **appropriée à chaque membre** de l'équipe
  - Choix technologiques
    - *« Nous voulons … BD-R avec JAVA » et les intervenants techniques pensent que BD-OO avec C++ est mieux adapté :* ***Il faut le proposer !***

L’art de la planification

- La planification **ne doit pas être coûteuse** et les objectifs sont de :
  - **Réunir** les membres de l’équipe
  - **Décider** du périmètre
  - **Émettre des estimations** de coûts et de délais
  - **Donner confiance** à chacun quant à la faisabilité du projet
  - **Fournir une référence** permettant d’évaluer le feedback

L’art de la planification

- Règles de la planification
  - Ne planifier que ce qui est nécessaire pour **atteindre le prochain objectif**
  - **Accepter** ses responsabilités
  - La personne **responsable** de l’implémentation est **habilitée** à faire l’estimation
  - **Ne pas tenir compte des dépendances entre les tâches**
  - **Un plan de priorités n’est pas un plan de développement**

L'art de la planification

- Déroulement
  - **Exploration** : on découvre les **nouvelles fonctionnalités** qui pourraient faire partie du système
    - *Écrire des scénarii client*
    - *Estimer les scénarii client et si besoin les découper*
  - **Engagement** : on décide d'un **sous-ensemble** de toutes les fonctionnalités possibles
    - *Trier par valeurs : jamais sans elle, pas essentielles mais significative, agréable à avoir*
    - *Trier par risque : estimation précise, précision raisonnable, aucune idée de l'estimation*
  - **Pilotage** : on guide le développement à mesure que **la réalité s'impose à la planification**
    - *Itération*

Développement

- **Règle d'or du développement** : on résout les problèmes du jour et c'est tout, les problèmes de demain seront résolus plus tard !
- Intégration ...
- Responsabilité ... du ...
- Programmation en ...

Tests

- Tests unitaires et fonctionnels
  - Les programmeurs
    - *Un test par méthode si :*
      - La signature **n'est pas absolument évidente**
      - Son implémentation sera **peut être complexe**
      - Il est possible de trouver **des cas bizarres**
      - Un **problème** apparaît
      - Le **refactoring** est nécessaire
  - Les clients
  - Autres tests
    - ***Parallèle*** *: migration d'un système vers un autre*
    - ***Charge*** *: quel est le comportement du logiciel*
    - ***Singe*** *: le système répond de façon sensée même si les saisies ne le sont pas*

Adopter XP

1. Déterminer quel est votre **pire** problème
2. Le **résoudre** selon la philosophie d'XP
3. Une fois que ce n'est plus votre pire problème, **recommencez**

Migrer vers XP

- Il est plus facile de faire migrer une équipe non constituée vers XP qu'une équipe existante
- Il est plus facile de faire migrer un projet qui commence vers XP qu'un projet en cours

XP

- Règles de 20-80 : **on obtient 80 % des bénéfices quand on n'a fait que 20 % du travail !**
- XP s'appuie sur cette règle : implémenter les meilleurs 20 % des fonctionnalités, mettre en place les meilleurs 20 % de la conception, appliquer la règle des 20-80 pour reporter l'optimisation

- Chacun **choisit** ses tâches
- Chacun **estime** ses tâches
- On procède au **rééquilibrage** si quelqu'un a trop de tâches

XP ne fait pas de miracle

- *« Même si les limites d'XP ne sont pas encore clairement définies, certains obstacles existent qui peuvent empêcher totalement sa pratique –des équipes de grande taille, des clients qui ne savent pas faire confiance, des technologies réfractaires au changement ou à l'élégance »*

Conclusion

- *« Toutes les méthodologies ont pour origine la peur*
- *J'ai peur*
  - *De faire du travail sans importance*
  - *De voir des projets annulés faute de décisions techniques de ma part*
  - *De prendre des décisions d'entreprise, et me planter*
  - *De voir des décideurs prendre des décisions techniques à ma place, et se planter*
  - *D'arriver à la fin d'une carrière passée à réaliser des systèmes, et m'apercevoir que j'aurais dû passer plus de temps avec mes enfants*
  - *De faire du travail dont je ne sois pas fier »*

Conclusion

- *« XP reflète aussi ce dont je n'ai pas peur*
  - *Coder*
  - *Changer d'avis*
  - *Avancer sans savoir tout ce que le futur me réserve*
  - *Faire confiance aux autres*
  - *Modifier la conception ou l'analyse d'un système opérationnel*
  - *Écrire des tests »*