Introduction aux Méthodes Agiles

1. **Pourquoi les méthodes Agiles ?** Avant l'agilité, les équipes utilisaient des méthodes **classiques** (cycle en V, waterfall) où :

- les besoins sont définis au début,
- le développement se fait ensuite,
- les tests arrivent à la fin.

Problème : si les besoins changent (ce qui arrive souvent), il faut tout recommencer.

➤ **Exemple simple**

Une équipe doit développer une application "Gestion de bibliothèque".
Dans le cycle en V :

- Les exigences sont fixées dès le début.
- Si la bibliothèque change ses besoins (ex : ajouter les livres numériques), c'est compliqué.

Les méthodes agiles permettent **d'adapter le projet en cours de route**.

2. **Définition des méthodes Agiles**

Les méthodes Agiles sont un ensemble de pratiques permettant de :

✓ développer **rapidement**,
✓ s'adapter aux **changements**,
✓ impliquer fortement le **client**,
✓ livrer un produit **progressif**.

3. **Les Principes Agile**

Il existe 12 principe nous citons les plus essentiels :

- Livrer rapidement et souvent.
- Accueillir les changements.
- Collaboration permanente avec le client.
- Communication directe (réunions courtes).
- Mesure principale : **logiciel fonctionnel**.
- Développement simple.
- Amélioration continue.

4. **Les principales méthodes Agiles**

4.1. SCRUM

5.1.1. Les rôles dans Scrum

Product Owner (PO)

Le **responsable du produit**.
Il représente le **client** et définit ce qui doit être développé.

Scrum Master (SM)

Le **garant de la méthode Scrum**.
Il n'est ni chef ni supérieur : il accompagne l'équipe.

Équipe de développement (Dev Team)

L'équipe qui **code, teste et conçoit** le produit.
Autonome et pluridisciplinaire.

5.1.2. Les événements Scrum

Sprint

Cycle de travail **fixe** : 1 à 4 semaines.
À la fin : une **version fonctionnelle** du produit.

Exemple :
Sprint de 2 semaines pour développer :

- Login
- Page de profil étudiant

Daily Meeting (Daily Scrum)

Réunion courte **de 15 minutes chaque jour**.

Chaque membre répond à :

1. Qu'ai-je fait hier ?
2. Que vais-je faire aujourd'hui ?
3. Quels obstacles rencontrés ?

Sprint Review

Réunion de **démonstration** à la fin du Sprint.

Objectifs :

- Montrer ce qui a été réalisé.

- Recevoir le feedback du Product Owner.
- Décider les ajustements pour les prochains Sprints.

Exemple :
L'équipe montre la page "Notes de l'étudiant" et le PO propose d'ajouter un filtre par module.

Sprint Rétrospective

Réunion interne de l'équipe après la Sprint Review.

Objectifs :

- Identifier ce qui a bien marché.
- Analyser ce qui doit être amélioré.
- Décider d'une action d'amélioration pour le prochain Sprint.

Exemple :
"Nous devons réduire le nombre de tâches en cours."
"Nous allons améliorer nos tests unitaires."

5.1.3. Les artefacts Scrum

Product Backlog

Une **liste de toutes les fonctionnalités** du produit, priorisées.

Contient des **User Stories**.

Exemple :

- US1 : En tant qu'étudiant, consulter les notes.
- US2 : En tant qu'enseignant, entrer les notes.
- US3 : En tant qu'administrateur, gérer les comptes.

Le PO est responsable du Backlog.

Sprint Backlog

Liste des éléments du Product Backlog **choisis pour le Sprint** + les tâches nécessaires.

Exemple :
Sprint 2 (2 semaines)

- Développer “Affichage des notes”
- Ajouter base de données
- Tests unitaires

Burndown chart

Graphique qui montre la **progression du Sprint**.

Il affiche :

- quantités de travail restant,
- jours du Sprint.

Objectif : vérifier si l’équipe est en avance, à l’heure ou en retard.

Exemple :
Si la courbe reste trop plate → le Sprint est en retard.

Exemple SCRUM : Application “Gestion pour étudiants”

Objectif : Une petite plateforme pour consulter les notes.

Sprint 1 (2 semaines) :

- Page de login
- Création d’un compte
- Interface simple étudiant

Sprint 2 :

- Import des notes par l'enseignant
- Affichage des notes

À chaque sprint, l’équipe **montre une version fonctionnelle**.

5.2. La méthode XP (Extreme Programming)

5.2.1. Introduction à XP

Extreme Programming (XP) est une méthode agile orientée principalement vers le **développement logiciel**. Elle met l’accent sur la **qualité du code**, l’**adaptation rapide aux changements** et la **collaboration étroite** avec le client.
XP est particulièrement adaptée aux projets où les besoins évoluent souvent ou sont difficiles à définir dès le départ.

5.2.2. Objectifs principaux de XP

- **Améliorer la qualité du logiciel** grâce à des pratiques de développement rigoureuses.
- **Réduire les risques** liés aux changements tardifs.

- **Favoriser une communication continue** entre l'équipe et le client.
- **Obtenir un produit fonctionnel** rapidement avec des livraisons fréquentes.

5.2.3. Les valeurs fondamentales de XP

XP repose sur **5 valeurs** de base :

1. ***Communication***

Les développeurs, testeurs et le client doivent communiquer régulièrement pour éviter les malentendus.

2. ***Simplicité***

Écrire le **code le plus simple possible** qui fonctionne, sans ajouter de fonctionnalités inutiles.

3. ***Feedback***

Recevoir rapidement des retours sur :

- le code (via tests, revues),
- le produit (via le client),
- les choix techniques.

4. ***Courage***

Avoir le courage :

- de simplifier,
- de refactoriser,
- de jeter du code inutile,
- de dire non quand nécessaire.

5. ***Respect***

Tous les membres de l'équipe sont valorisés et impliqués dans les décisions.

5.2.4. Les pratiques clés de XP

XP est connue pour ses **pratiques techniques très structurées**, souvent poussées "à l'extrême", d'où le nom.

5.2.4.1. Développement piloté par les tests (TDD)

- Écrire le test **avant** le code.
- Le test échoue → écrire le code minimal → test réussi → refactoriser.

5.2.4.2. Programmation en binôme (Pair Programming)

Deux développeurs travaillent sur **un même poste** :

- l'un écrit le code,
- l'autre contrôle et réfléchit à l'amélioration.

5.2.4.3. Intégration continue

Le code doit être :

- intégré plusieurs fois par jour,
- testé automatiquement,
- compilé sans erreurs.

5.2.4.4. Refactoring

Amélioration continue du code :

- simplification,
- amélioration des performances,
- suppression de doublons.

5.2.4.5. Propriété collective du code

Tout membre de l'équipe peut modifier n'importe quelle partie du code.
Encourage la responsabilité et la qualité globale.

5.2.4.6. Standards de codage

Tous les développeurs doivent suivre les **mêmes conventions**, pour maintenir un code homogène.

5.2.4.7. Livraisons fréquentes

Petits incréments livrés régulièrement (toutes les 1–3 semaines).

5.2.4.8. Présence du client (On-site Customer)

Un **représentant du client** doit être disponible en permanence pour :

- clarifier les exigences,
- valider rapidement les fonctionnalités,
- prioriser les besoins.

5.2.5. Cycle de développement XP

XP utilise un processus itératif, avec des cycles courts :

1. **Écouter** → comprendre les besoins du client.

2. **Planifier** → définir les user stories et priorités.
3. **Concevoir / Coder** → en suivant TDD, pair programming, refactoring…
4. **Tester** → tests unitaires et tests d'acceptation.
5. **Livrer** → petit incrément fonctionnel.
6. **Améliorer** → rétrospective et ajustements.

Remarque : **TDD** signifie **Test-Driven Development** (Développement guidé par les tests). C'est une pratique agile où **on écrit d'abord les tests avant d'écrire le code**.

5.2.6. Avantages de XP

- Très **forte qualité du code**.
- Adaptation rapide aux changements.
- Communication fluide avec le client.
- Risques techniques réduits.
- Tests automatisés garantissant la stabilité.

5.2.7. Limites de XP

- Nécessite une **forte discipline** de l'équipe.
- Demande un client **disponible en permanence**.
- Pair programming peut être coûteux.
- Peut être difficile à appliquer dans de grandes équipes.

5.2.8. Exemple simple d'application XP

Contexte : Développement d'un module "Authentification"

Étape 1 – User Story

- *"En tant qu'utilisateur, je veux me connecter avec email et mot de passe."*

Étape 2 – TDD

- Écrire un test :
  TestLogin : vérifier qu'un utilisateur valide accède au système.

Étape 3 – Code minimal

- Implémenter la fonction login(email, password) pour passer le test.

Étape 4 – Refactoring

- Nettoyer le code (simplification, noms plus clairs…).

Étape 5 – Livraison

- Livraison d'un petit module utilisable.

Étape 6 – Feedback

- Le client teste et demande éventuellement une évolution (ex : ajout vérification captcha).

5.3. Kanban

Méthode visuelle utilisant un tableau avec 3 colonnes :

| À faire | En cours | Terminé |

On limite le nombre de tâches "En cours" pour éviter d'être surchargé.

Exemple Kanban :

Développement du site "Club Informatique".

- Carte 1 : Créer page d'accueil → *En cours*
- Carte 2 : Ajouter galerie photos → *À faire*
- Carte 3 : Ajouter formulaire contact → *Terminé*

6. Cycle de vie d'un projet Agile

1. **Expression générale du besoin**
2. "Une app pour gérer les inscriptions des étudiants."
3. **Création du Product Backlog**
   Liste des fonctionnalités :
   - Authentification
   - Inscription aux modules
   - Consultation du planning
   - Notifications…
4. **Planification du Sprint** (durée : 1 à 4 semaines)
5. **Développement**
   Travail par petites portions.
6. **Daily Scrum** (réunion 15 minutes)
   - Ce que j'ai fait hier
   - Ce que je vais faire aujourd'hui
   - Les obstacles
7. **Livraison du Sprint**
   On montre la version fonctionnelle.
8. **Rétrospective**
   L'équipe discute :
   - Ce qui a bien marché
   - Ce qu'on doit améliorer

7.Conclusion

Les méthodes agiles permettent :

- une gestion plus souple,
- une meilleure communication,

- une livraison rapide,
- une réduction des risques.

SCRUM est la méthode la plus utilisée en entreprise.