

# Présentation XP

## Introduction

Les piliers de l'eXtreme Programming (XP) reposent sur des mécanismes clés pour :

- Optimiser l'efficacité des équipes (pair programming, intégration continue, etc.),
- Raccourcir les boucles de feedback (livraisons fréquentes, tests automatisés, rétrospectives),
- Fluidifier les releases (déploiement continu, petites itérations incrémentales),
- Renforcer la collaboration client (user stories, planification adaptative, démonstrations régulières).

La maîtrise de XP passe par une compréhension hiérarchisée :

- Ses Valeurs (communication, simplicité, feedback, courage, respect) → fondations culturelles.
- Ses Principes (ex : "Embrasser le changement", "Livrer de la valeur rapidement") → cadrage stratégique pour choisir/adapter les pratiques.
- Ses Pratiques (TDD, refactoring, CI/CD, velocity tracking, etc.) → outils concrets pour implémenter les principes.

Adaptation contextuelle :

XP n'est pas un framework rigide. Certaines pratiques (ex : pair programming) peuvent être ajustées ou combinées (avec Scrum/Kanban) selon :

- La taille de l'équipe (startup vs. scale-up),
- La criticité du projet (MVP vs. système embarqué),
- La maturité technique (legacy code vs. greenfield).

XP Agile DevOps TDD Clean code

## Valeurs XP

Les **valeurs XP (eXtreme Programming)** définissent la culture et les comportements clés pour des équipes **agiles et techniques**.

### Tableau des Valeurs XP

Valeur	Définition Technique
Communication	<b>Échanges structurés</b> pour :
	- *Knowledge sharing* : <b>mob programming</b> , docs collaboratives (ex : Confluence/Notion).
	- *Résolution de problèmes* : <b>stand-ups techniques</b> , canaux dédiés (Slack/Teams).
	- *Coordination* : <b>planning poker</b> , raffinement de backlog.
Simplicité	<b>Approche minimaliste</b> :
	- *YAGNI* : <b>"You Aren't Gonna Need It"</b> → pas de sur-ingénierie.
	- *DTSTTCPW* : <b>"Do The Simplest Thing That Could Possibly Work"</b> .
	- *Refactoring* : <b>incrémental</b> (ex : *boy scout rule*).
Feedback	<b>Boucles courtes et automatisées</b> :
	- *Tests* : <b>TDD/BDD</b> (feedback en ms).
	- *Demos* : <b>Sprint reviews</b> (validation client en jours).
	- *Rétros* : <b>Amélioration continue</b> des processus.

Valeur	Définition Technique
Courage	<b>Décisions techniques audacieuses :</b>
	- Remettre en question les choix (ex : "Ce framework est-il adapté ?").
	- *Fail fast* : <b>Spikes</b> pour valider des hypothèses avant le dev.
Respect	- *Blameless culture* : <b>Postmortems</b> sans jugement.
	<b>Collaboration bienveillante :</b>
	- *Pair programming* : <b>Montée en compétence</b> (juniors/seniors).
	- *Code reviews* : <b>Focus sur l'amélioration</b> , pas la critique.
	- *Retros actionables* : <b>Écoute active</b> des feedbacks.

## Principes XP - Fondations pour les Pratiques Agiles

Les **principes XP** servent de **pont entre les valeurs et les pratiques**. Ils expliquent **pourquoi** certaines pratiques (comme le TDD ou le pair programming) sont efficaces, et comment les adapter à différents contextes techniques.

### Tableau des Principes XP

Principe	Définition Technique	Applications Concrètes	Outils/Exemples
Humanity	<b>Bien-être et croissance des devs :</b>	→ <b>Réduit le turnover</b> (équipes stables = moins de *knowledge loss*).	- <b>1:1s réguliers</b> (feedback constructif).
	- Sécurité psychologique (*psychological safety*) pour oser poser des questions.	→ <b>Améliore la collaboration</b> (moins de *silos*).	- <b>Retrospectives anonymes</b> (ex : Toolbox).
	- Satisfaction via l'autonomie et l'impact visible du travail.	→ <b>Boost la productivité</b> (devs engagés = moins de *procrastination*).	- <b>Tableaux de reconnaissance</b> (ex : kudos Slack).
Economics	<b>Alignement avec la valeur métier :</b>	→ <b>Priorise les features à fort ROI</b> (évite le *waste*).	- <b>User Story Mapping</b> (ex : Miro).
	- Toute décision technique doit être évaluée en termes de <b>coût vs. valeur business</b> .	→ <b>Optimise les coûts d'infrastructure</b> (ex : serverless vs. bare metal).	- <b>Coût de la dette technique</b> (ex : SonarQube).
	- Exemple : *"Ce refactor vaut-il 2 sprints si le gain métier est nul ?"*.	→ <b>Justifie les investissements techniques</b> (ex : migration vers Kubernetes).	- <b>Business Case Templates</b> .
Mutual Benefit	<b>Pratiques gagnant-gagnant</b> pour l'équipe <b>et</b> le business :	→ <b>Code plus maintenable</b> = moins de *firefighting*.	- <b>TDD</b> (JUnit, pytest).
	- Exemples :	→ <b>Livraisons plus fréquentes</b> = feedback client plus rapide.	- <b>CI/CD</b> (GitHub Actions, GitLab CI).
	- *Tests automatisés* : gain temps pour les devs <b>et</b> qualité pour le client.	→ <b>Réduction des bugs en prod</b> = moins de *hotfixes*.	- <b>Clean Code</b> (ESLint, Prettier).
	- *Code simple et lisible* : maintenabilité <b>et</b> onboarding facilité.		

Principe	Définition Technique	Applications Concrètes	Outils/Exemples
Self-similarity	<b>Réutilisation des solutions</b> avec adaptation contextuelle :	→ <b>Accélère le développement</b> (moins de *reinventing the wheel*).	- <b>Design Patterns</b> (ex : Strategy, Adapter).
	- Un pattern qui marche dans un micro-service peut inspirer une solution dans un autre.	→ <b>Standardise les bonnes pratiques</b> (ex : conventions de nommage).	- <b>Architecture Decision Records</b> (ADR).
	- <b>Mais</b> : chaque contexte est unique (ex : *legacy code* vs. *greenfield*).	→ <b>Évite les *cargo cults*</b> (copier-coller sans comprendre).	- <b>Spikes techniques</b> pour valider l'adéquation.
Improvement	<b>Amélioration continue</b> (kaizen) via des itérations courtes :	→ <b>Qualité incrémentale</b> (ex : couverture de tests +1% par sprint).	- <b>TDD</b> (red-green-refactor).
	- Pas de perfectionnisme, mais des <b>petites améliorations régulières</b> .	→ <b>Processus optimisés</b> (ex : réduction du *lead time*).	- <b>Rétrospectives</b> (Mad/Sad/Glad).
	- Exemple : refactoring de 10% du code à chaque PR.	→ <b>Culture d'apprentissage</b> (ex : *blameless postmortems*).	- <b>Metrics DevOps</b> (DORA).
Diversity	<b>Hétérogénéité des profils</b> comme force :	→ <b>Solutions plus innovantes</b> (ex : approche *junior* vs. *senior*).	- <b>Pair Programming</b> (rotatif).
	- Compétences complémentaires (ex : *backend* + *DevOps*).	→ <b>Moins de *groupthink*</b> (biais de conformité).	- <b>Workshops cross-fonctionnels</b> .
	- Respect des opinions divergentes (ex : *"Pourquoi pas Serverless ?"*).	→ <b>Meilleure résolution de problèmes</b> (ex : *brainstorming technique*).	- <b>Tools collaboratifs</b> (Miro, Mural).
Reflection	<b>Analyse post-action</b> pour capitaliser sur les expériences :	→ <b>Évite de répéter les mêmes erreurs</b> (ex : *post-incident reviews*).	- <b>Rétrospectives</b> (Start/Stop/Continue).
	- Exemples :	→ <b>Améliore les processus</b> (ex : ajustement des *DoD*).	- <b>Journaux techniques</b> (ex : Notion, Confluence).
	- <b>*Rétrospectives*</b> : "Pourquoi ce bug a-t-il passé les tests ?".	→ <b>Renforce la résilience d'équipe</b> .	- <b>Metrics de qualité</b> (ex : Sentry pour les erreurs).
	- <b>*Code reviews*</b> : "Pourquoi ce PR a pris 3 jours ?".		
Flow	<b>Fluidité du travail</b> pour minimiser les interruptions :	→ <b>Réduit le *cycle time*</b> (ex : de l'idée à la prod).	- <b>Kanban</b> (Jira, Trello).
	- Exemples :	→ <b>Limite le *multitasking*</b> (moins de *context switching*).	- <b>WIP Limits</b> .
	- <b>*Intégration continue*</b> : feedback en minutes.	→ <b>Livraisons plus prévisibles</b> .	- <b>CI/CD pipelines</b> .
	- <b>*Petites user stories*</b> : évite les blocages longs.		
Opportunity	<b>Transformer les problèmes en leviers</b> :	→ <b>Innovation technique</b> (ex : migration vers un nouveau langage).	- <b>Spikes d'exploration</b> .
	- Exemples :	→ <b>Amélioration des compétences</b> (ex : formation sur un nouveau tool).	- <b>Hackathons internes</b> .
	- Un <b>*bug critique*</b> → opportunité pour améliorer les tests.	→ <b>Renforce la cohésion d'équipe</b> (résolution collaborative).	- <b>Blameless Postmortems</b> .
	- Une <b>*dette technique*</b> → occasion de refactor.		

Principe	Définition Technique	Applications Concrètes	Outils/Exemples
<b>Redundancy</b>	<b>Duplication utile</b> (vs. *waste*) :	→ <b>Résilience du système</b> (ex : réplicas de bases de données).	- <b>Backup automatiques.</b>
	- Exemples :	→ <b>Sécurité</b> (ex : double vérification des PRs).	- <b>Linters/Formatters</b> (redondance des checks).
	- *Tests redondants* : couverture multi-niveaux (unitaires + intégration).	→ <b>Onboarding facilité</b> (ex : docs répétées dans plusieurs formats).	- <b>Documentation as Code</b> (ex : Markdown + CI).
	- *Pair programming* : 2 paires d'yeux sur le même code.		
<b>Failure</b>	<b>Apprentissage via l'échec</b> :	→ <b>Culture de l'expérimentation</b> (ex : *feature flags* pour tester en prod).	- <b>Feature Toggles</b> (LaunchDarkly).
	- Exemples :	→ <b>Solutions plus robustes</b> (ex : *chaos engineering*).	- <b>Chaos Monkey</b> (Netflix).
	- Un *déploiement raté* → amélioration du pipeline CI/CD.	→ <b>Équipe plus résiliente</b> (moins de peur de l'échec).	- <b>Postmortem Templates.</b>
	- Un *bug en prod* → renforcement des tests E2E.		
<b>Quality</b>	<b>Non-négociable</b> : la qualité est un <b>multiplicateur de productivité</b> .	→ <b>Moins de *technical debt*</b> (ex : code propre = moins de *firefighting*).	- <b>SonarQube</b> (qualité de code).
	- Exemples :	→ <b>Meilleure vélocité à long terme</b> (moins de régressions).	- <b>TDD/BDD.</b>
	- *Clean Code* : noms de variables explicites, fonctions courtes.	→ <b>Fierté du travail</b> (motivation accrue).	- <b>Code Reviews strictes.</b>
	- *Tests automatisés* : 100% de couverture pour le code critique.		
<b>Baby Steps</b>	<b>Petites itérations</b> pour éviter les erreurs coûteuses :	→ <b>Réduit les risques</b> (ex : *big bang refactor* → *incremental changes*).	- <b>TDD</b> (petits tests → petit code).
	- Exemples :	→ <b>Livraisons plus fréquentes</b> (ex : *trunk-based development*).	- <b>Feature Branches courtes.</b>
	- *Test-First Programming* : écrire un test avant le code.	→ <b>Feedback immédiat</b> (ex : *red-green-refactor*).	- <b>Git hooks</b> (pre-commit tests).
	- *Petites PRs* : max 200 lignes de code.		
<b>Accepted Responsibility</b>	<b>Responsabilité active</b> (vs. assignée) :	→ <b>Ownership accru</b> (ex : un dev *possède* une feature de bout en bout).	- <b>User Story Assignment</b> (volontariat).
	- Exemples :	→ <b>Qualité accrue</b> (ex : *"Si je code la feature, je teste aussi ses edge cases"*).	- <b>Definition of Done (DoD) stricte.</b>
	- *TDD* : le dev qui écrit le code écrit aussi les tests.	→ <b>Moins de *handovers* problématiques.</b>	- <b>Pair Programming</b> (responsabilité partagée).
	- *On-call* : rotation volontaire pour la prod.		

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/eadi/bloc3/xp/chap1?rev=1763896911>

Last update: **2025/11/23 12:21**

