

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

Valeur	Définition Technique
Courage	Décisions techniques audacieuses :
	- Remettre en question les choix (ex : "Ce framework est-il adapté ?").
	- *Fail fast* : Spikes pour valider des hypothèses avant le dev.
	- *Blameless culture* : Postmortems sans jugement.
Respect	Collaboration bienveillante :
	- *Pair programming* : Montée en compétence (juniors/seniors).
	- *Code reviews* : Focus sur l'amélioration , pas la critique.
	- *Retros actionables* : Écoute active 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	Bien-être et croissance des devs :	→ Réduit le turnover (équipes stables = moins de *knowledge loss*).	- 1:1s réguliers (feedback constructif).
	- Sécurité psychologique (*psychological safety*) pour oser poser des questions.	→ Améliore la collaboration (moins de *silos*).	- Retrospectives anonymes (ex : Toolbox).
	- Satisfaction via l'autonomie et l'impact visible du travail.	→ Boost la productivité (devs engagés = moins de *procrastination*).	- Tableaux de reconnaissance (ex : kudos Slack).
Economics	Alignement avec la valeur métier :	→ Priorise les features à fort ROI (évite le *waste*).	- User Story Mapping (ex : Miro).
	- Toute décision technique doit être évaluée en termes de coût vs. valeur business .	→ Optimise les coûts d'infrastructure (ex : serverless vs. bare metal).	- Coût de la dette technique (ex : SonarQube).
	- Exemple : *"Ce refactor vaut-il 2 sprints si le gain métier est nul ?"*	→ Justifie les investissements techniques (ex : migration vers Kubernetes).	- Business Case Templates .
Mutual Benefit	Pratiques gagnant-gagnant pour l'équipe et le business :	→ Code plus maintenable = moins de *firefighting*.	- TDD (JUnit, pytest).
	- Exemples :	→ Livraisons plus fréquentes = feedback client plus rapide.	- CI/CD (GitHub Actions, GitLab CI).
	- *Tests automatisés* : gain temps pour les devs et qualité pour le client.	→ Réduction des bugs en prod = moins de *hotfixes*.	- Clean Code (ESLint, Prettier).
	- *Code simple et lisible* : maintenabilité et onboarding facilité.		

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

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

