Introduction à l'eXtreme Programming et au développement agile

Gauthier Picard

SMA/G2I/ENS Mines Saint-Etienne

gauthier.picard@emse.fr

Octobre 2009

Adapté de *XP ou les bienfaits d'un développement « agile »*

par Kévin Ottens, KDE Core Developer

Sommaire

Sommaire

La préhistoire : Influences

Culture SmallTalk

- 198x : Kent Beck et Ward Cunningham travaillent chez Tektronix
- Elements clef de cette culture :
  - Développement en binôme
  - Refactoring
  - Changements rapides
  - Interaction permanente avec le client
  - Intégration continue
  - Développement itératif
  - Tests permanents

« Episodes »

- 1986-1996 : Kent Beck et Ward Cunningham développent un ensemble de bonnes pratiques
- Support dans le language Episodes de Ward Cunningham

La préhistoire : Influences (cont.)

Refactoring

- 1989-1992 : William F. Opdyke, "Refactoring Object-Oriented Frameworks". PhD Thesis
- 1995-1996 : Kent Beck, "Smalltalk Best Practices Patterns"
- 1999 : Martin Fowler, "Refactoring : Improving the Design of Existing Code"

Test-Driven Development

- Dérivé des pratiques de refactoring
- Premier article publié par Kent Beck à propos de SmallUnit
- 1997 : Création de JUnit par Kent Beck et Erich Gamma

La préhistoire : Influences (cont.)

Design Patterns

- Idée reprise de l'architecture, plus particulièrement de Christopher Alexander
- Kent Beck et Ward Cunningham ont appliqué le concept de patrons au logiciel depuis 1987
- 1995 : Publication de *Design Patterns*, par
  - Erich Gamma
  - Richard Helm
  - Ralph Johnson
  - John Vlissides

Les débuts

Projet C3 (Chrysler, 1996) : Naissance du processus

- Intervention de Kent Beck pour améliorer les performances
- Identification de maux plus profonds
- La direction lui confie l'équipe pour récrire le logiciel
- Mise au point des pratiques XP avec Ron Jeffries
  - Correspond à la somme des influences vues précédemment
  - Octobre 1998 : Article dans *Distributed Computing* magazine sur C3 et XP

De l'utilisation du Wiki Wiki Web

- Ward Cunningham a mis au point ce site collaboratif
- Utilisé par la communauté XP pour affiner et discuter le processus

Les débuts (cont.)

Evolution et gain de popularité

- Septembre 1999 : "Extreme Programming Explained" par Kent Beck
- Processus presque identique à celui appliqué sur C3
- Intégration du processus Scrum pour la gestion de projet
- Décembre 1999 : 1^{e} "XpImmersion" à ObjectMentor
- 31 Décembre 1999 : Création du groupe de discussion XP sur *Yahoo !*
- Juin 2000 : 1^{e} conférence internationale sur XP
- Septembre 2000 : "Extreme Programming Installed" écrit entre autre par Ron Jeffries
- Juillet 2001 : Première conférence "XpUniverse"

Le Manifeste Agile

- Réunion organisée par Kent Beck en Février 2001
- 17 personnalités, dont les créateurs de Crystal, Scrum, Adaptive Software Development, etc.

http://www.agilemanifesto.org/

Nous cherchons de meilleures manières pour développer des logiciels en aidant les autres et en développant nous mêmes. À travers ce travail nous en sommes venus à valoriser :

Personnes et interaction plutôt que processus et outils
Logiciel fonctionnel plutôt que documentation complète
Collaboration avec le client plutôt que négociation de contrat
Réagir au changement plutôt que suivre un plan

En fait, bien que les éléments de droite soient importants, nous pensons que les éléments de gauche le sont encore plus.

Mythe des phases

Processus séquentiels

- La plupart dérivés du cycle en V
- Variantes... intégrant parfois des sous-cycles itératifs

1. Cahier des charges
2. Document de spécifications
3. Document de conception générale
4. Document de conception détaillée
5. Plans de tests
6. …
7. Et l'application ?

Mythe des phases

Processus séquentiels

- La plupart dérivés du cycle en V
- Variantes... intégrant parfois des sous-cycles itératifs

- 1994 : Rapport CHAOS (étude sur 8000 projets)
  - 16% finalisés dans les temps et le budget prévus
  - 32% interrompus en cours de route
- Diagnostic courant des dérapages :
  - Spécifications ambitieuses et Conception poussée
  - Puis, enlisement de la construction
    - Remise en cause de choix initiaux
    - Mise à mal de la conception proposée
- Effet Tunnel
- Le processus correspond il aux besoins du projet ?

Utopie des spécifications immuables

Faits

- Sources principales de problèmes dans les spécifications :
  - Erreurs
  - Oublis
  - Changements
- Définir un logiciel *a priori* est un exercice difficile, sauf dans le cas :
  - D'applications extrêmement simples
  - De rares contextes connus et maitrisés
- Le client a du mal à imaginer ce que sera l'application

Conséquences

- Remises en question des spécifications
- Risques pour la conception liés au changement

Valeurs XP

Communication

- Développement = Effort collectif de création
  - Avoir une vision commune et pouvoir se synchroniser
  - → Qualité de la communication
- Communication directe et le contact humain
  - Faiblesse pour la traçabilité et la structuration
  - Augmentation de la réactivité
- Communication écrite présente, en général par du code

Simplicité

- « La chose la plus simple qui puisse marcher »
- Simple ≠ Simpliste
- Eviter la complexité inutile dans le code
- Toute duplication doit être éliminée

Valeurs XP (cont.)

Retour d'information (Feedback)

- Boucles de feedback pour réduire les risques
  - Connaître l'état du projet
  - Rectifier le tir si nécessaire
- Facteur de qualité
  - Acquisition d'expérience
  - Amélioration constante du travail

Courage

- Se lancer dans un projet non entièrement spécifié
- Accepter de remanier une portion de code devenue complexe
- Appliquer les valeurs de feedback et de communication
  - Accepter de montrer ses propres limites
  - Maintenir une transparence complète

Principes XP

- **Le client (maîtrise d'ouvrage) pilote lui-même le projet**, et ce de très près grâce à des cycles itératifs extrêmement courts (1 ou 2 semaines)
- **L'équipe livre très tôt dans le projet une première version** du logiciel, et les livraisons de nouvelles versions s'enchaînent ensuite à un rythme soutenu pour obtenir un feedback maximal sur l'avancement des développements
- **L'équipe s'organise elle-même pour atteindre ses objectifs**, en favorisant une collaboration maximale entre ses membres
- **L'équipe met en place des tests automatiques pour toutes les fonctionnalités** qu'elle développe, ce qui garantit au produit un niveau de robustesse très élevé
- **Les développeurs améliorent sans cesse la structure interne du logiciel** pour que les évolutions y restent faciles et rapides

Sommaire

Programmeur

Deux hypothèses dans XP pour justifier l'importance du code

- En génie logiciel, l'activité correspondant à la fabrication est la compilation, **pas** la programmation
- Code clair, structuré et simple modifié autant de fois que nécessaire pour qu'il soit compréhensible et non redondant est la meilleure forme de conception

Tests et proximité du client

- Tests (cf. Feedback)
  - Eviter les décalages entre ce que l'on veut coder et le comportement réel
- Ce qui n'est pas testé n'existe pas (cf. Courage)
- Ecouter le client (cf. Communication)
  - Lui seul sait comment le logiciel doit fonctionner

Programmeur (cont.)

Conception pour un codage durable

- Elle est très importante !
- Elle a un but différent :
  - Pas montrer ce que l'on a codé
  - Pas fournir des documents remplis de schémas
  - Garantir le long terme
- Eviter ainsi :
  - Fragilité (modification → régression)
  - Rigidité (petite modification → nombreuses modifications)
  - Immobilité (factoriser → tout casser)

Interventions sur la conception

- Collectivement lors des séances de planification
- Lors de l'écriture des tests unitaires
- Par le remaniement (refactoring)

Programmeur (cont.)

Responsabilisation

- Retour du programmeur comme rôle central
- A la fois : codeur, testeur, concepteur et analyste
- Apprentissage → Qualités humaines nécessaires
- XP : Ecole de l'excellence
- Responsabilisés pour donner le meilleur d'eux même
  - ex. : estimation des charges et délais

Pratiques XP

- Programmation en binôme
- Tests unitaires
- Conception simple
- Remaniement
- Responsabilité collective du code
- Règles de codage
- Intégration continue
- Rythme durable

Client

Qui est-il ?

- Pas nécessairement le client contractuel
  - Assistant à maîtrise d'ouvrage
  - Représentant des utilisateurs
- A défaut quelqu'un pour agir comme client « artificiel »
  - Chef de projet
  - Ingénieur chargé des spécifications

Client sur site et feedback

- Intégré à l'équipe de développement
- Plus de cahier des charges vague ou incompréhensible
- Plus de démo truquée
- Explique se qu'il souhaite aux développeurs
  - → Vision plus rapide du résultat
  - → Prise de conscience en cas d'erreur

Client (cont.)

Scénarios clients

- Description informelle d'une fonctionnalité ou d'une interaction avec l'utilisateur
- Le plus simple possible

Démarrage du projet

- Des scénarios initiaux sont dégagés et présentés
- Ils sont classés par priorité et répartis en itérations

A chaque itération...

- Grâce au feedback introduit (logiciel fonctionnel) :
  - Il peut revoir le contenu des itérations
  - Modifier ses scénarios
- Il est garant des fonctionnalités du logiciel

Client (cont.)

Tests de recette

- But : préciser les contours des scénarios
- Données concrètes levant les ambiguïtés
- Preuve que le système fait ce qu'il demandait
- A chaque fin d'itération tous les tests de recette doivent passer avec succès

Pratiques XP

- Planification itérative
- Rédaction des scénarios clients
- Séances de planification
- Tests de recette
- Intégration continue
- Rythme durable

Testeur

Le bras droit du client

- Définit et automatise les tests de recette
  - Conseille le client sur la testabilité d'une fonctionnalité
  - Utilisation d'outils différents pour scripter
- Garant du sentiment de réussite sur le projet
  - Test fonctionnel réussi → Progression
  - Communiquer pour motiver (graphique de progression...)

Compétences requises

- Programmeur hétéroclite (maîtriser l'outillage de test)
- Rigoureux et intègre

Pratiques XP

- Suivi des tests (planification itérative)
- Tests de recette
- Intégration continue
- Rythme durable

Tracker

Missions

- Suivre les tâches en cours d'itération
- Interroger les programmeurs
  - Savoir où ils en sont
  - Ne pas les mettre sur des charbons ardents
  - Attention à ne pas dériver en discussion technique
- Détecter les problèmes avant qu'il ne soit trop tard
  - Révélateur
  - Pas de prise d'initiative
- *Il fait en sorte que la tâche de 3 jours en prenne 4 et non 6*

Tracker (cont.)

Qui est-il ?

- Pas un supérieur hiérarchique
- Quelqu'un a qui on peut se confier
- De préférence pas un programmeur, mais quelqu'un d'extérieur
  - Pour éviter les discussions techniques
  - A défaut, ce rôle peut tourner entre les programmeurs à chaque itération

Pratiques XP

- Planfication itérative

Manager

Position dans l'organisation

- Supérieur hiérarchique des programmeurs
- Ne fait pas partie intégrante de l'équipe
- Chef du service auquel appartient l'équipe
- Chef de projet global (dans le cadre d'un sous-projet)

Responsabilités

- Il s'occupe matériellement de l'équipe
- Il demande des comptes (sur les engagements pris)
- Interface avec l'extérieur (dans le cadre d'un sous-projet)

Pratiques XP

- Scénarios client
- Planification itérative
- Rythme durable

Coach

Garant du processus

- Il réunit tout les autres rôles
- Vérifie que chaque rôle est respecté
- Ses buts ultimes :
  - Equipe autonome
  - Chaque programmeur est en amélioration continue
- Ses qualités
  - Expert de la méthode XP
  - Expert technique
  - Programmeur chevronné
  - Architecte
  - Pédagogue et sensible
  - Sang-froid

Pratiques XP

- Toutes !

Répartitions des rôles

Plusieurs rôles pour une personne

- Attention aux combinaisons possibles
- Toutefois, pas de règle absolue
- S'assurer qu'une cumulation ne pousse pas à sacrifier une composante importante d'un rôle

Plusieurs personnes pour un rôle

- Programmeur, le plus grand nombre
- Tracker, une seule personne... à un moment donné
- Coach, une personne unique
- Manager, une personne unique
- Client+Testeur, d'une personne à une équipe

Compatibilité des rôles

	Programmeur	Client	Programmeur	Tracker	Manager	Coach
Programmeur		✗	!	!	✗	!
Client	✗		✓	✗	✗	✗
Testeur	!	✓		✗	✗	
Tracker	!	✗	✗		!	!
Manager	✗	✗	✗	!		
Coach	!	✗	✗	!	✗	

✓ bonne combinaison

✗ mauvais combinaison

! combinaison envisageable mais risquée

Précautions

Comportements contre-indiqués

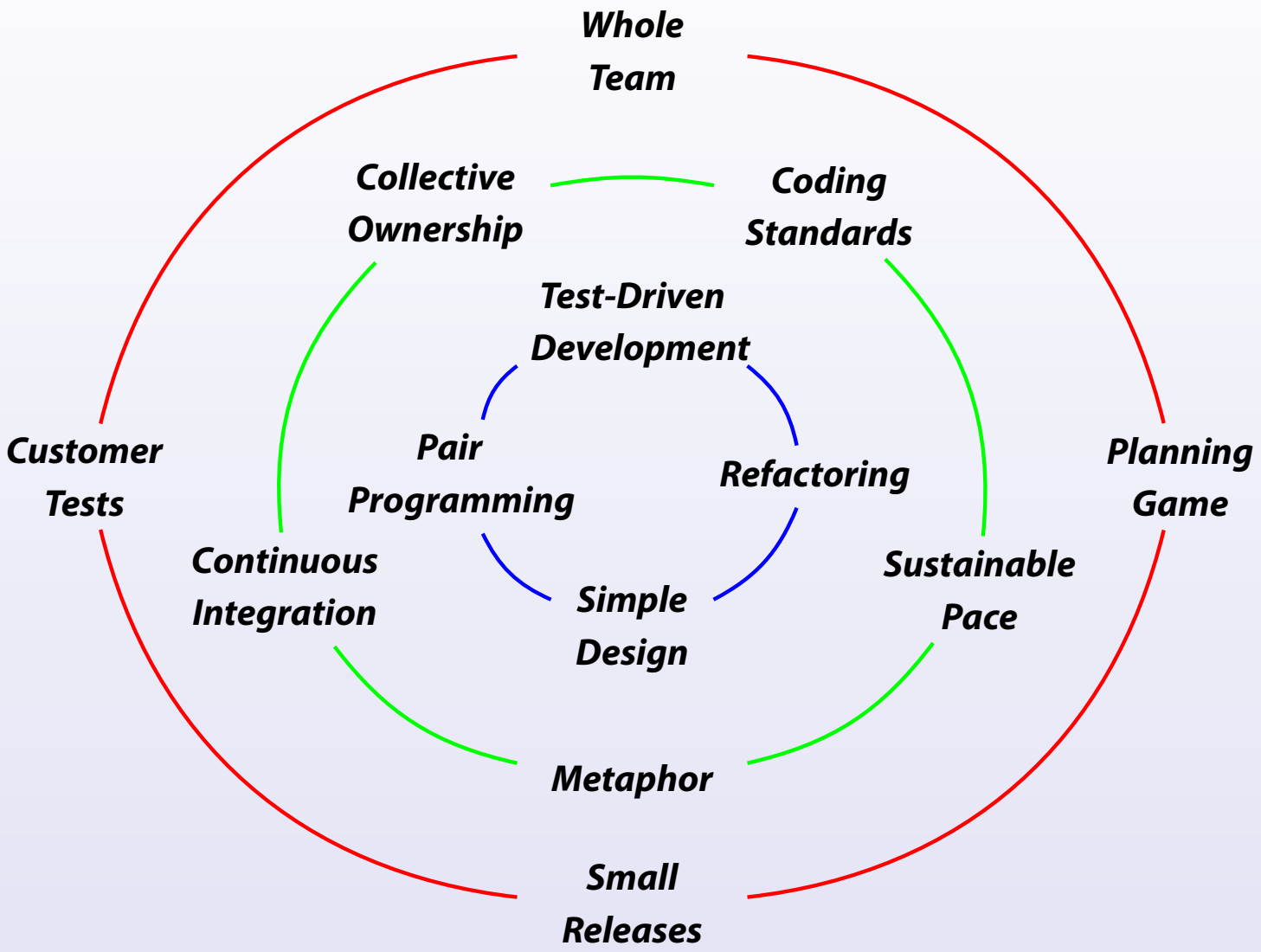
- S'attribuer les mérites ou rejeter la faute sur les autres
- Un codeur (même très compétent)
  - faisant des choses que personne ne comprend
  - refusant de travailler avec quelqu'un de moins compétent
- Chercher à se rendre indispensable

Eviter la spécialisation

- Tendance naturelle à la constitution de sous-groupes
  - Avec XP, pas de conséquences tant que le groupe est petit
  - Envisager des mécanismes de rotation pour les binômes
- Cas particulier du consultant
  1. Le consultant est appelé pour un problème précis
  2. Il travaille toujours accompagné par un programmeur
  3. L'équipe souhaitera probablement refaire le travail

Sommaire

Pratiques XP



Pratiques XP

Programmation	Développement piloté par les tests Conception simple Remaniement
Collaboration	Programmation en binôme Responsabilité collective du code Règles de codage Intégration continue
Gestion de projet	Client sur site Rythme durable Livraisons fréquentes Planification itérative

Développement piloté par les tests

Automatisation

- Tests d'un logiciel sont généralement automatisables
- Automatisation → moins de temps consacré aux tests
- Temps nécessaire à l'automatisation plus long, mais...
  - Moins de défauts dans le code
  - Moins de régressions

Ecrire les tests en premier

- Position a priori paradoxale, mais...
  - Elimination du dilemme habituel en fin de projet
  - Code plus facilement testable
  - Moins d'enchevêtrement → Meilleure conception
- Tests fonctionnels → Forme de spécification
- Tests unitaires → Forme de conception détaillée

Développement piloté par les tests (cont.)

Tests Fonctionnels

- Dans le cas général, la recette permet de vérifier que :
  - Le logiciel n'a pas d'anomalie constatée
  - Les fonctionnalités livrées sont bien celles demandées
- Dans le cas XP :
  - Tests de recettes spécifiés par le client
  - Tests fonctionnels écrits par le testeur
  - → Livraison possible lorsque tous les tests réussissent

Test Unitaires

- Ecrits avant le code à tester
- Plusieurs rôles :
  - Rythmer la programmation (progression « d'alpiniste »)
  - Guider la conception
  - Documenter le code produit (cf. notion de « contrat »)

Développement piloté par les tests (cont.)

Test Unitaires (nouvelle implémentation)

1. Identifier une sous-partie du problème
2. Ecrire un test indiquant si le problème est résolu
3. Exécuter le test... qui doit échouer
4. Ecrire le code
5. Exécuter le test pour vérifier que le code fonctionne correctement

Test Unitaires (modification de conception)

1. Modifier le test concerné
2. Exécuter le test... qui doit échouer
3. Modifier le code
4. Exécuter le test pour vérifier que le code fonctionne correctement

Conception simple

Eviter la spéculation

- Conception = Investissement (en temps et/ou complexité)
  - Identifier toutes les classes ou modules dont sera constitué le système, héritage...
  - Implémenter un système générique pour facilement implémenter chaque fonctionnalité spécifique
- Cet investissement peut-être bénéfique...
  - Mais si on a mal « spéculé » ?
- XP : Se concentrer sur une et une seule fonctionnalité
  - Obtenir la meilleure conception possible... ...
    - sans aller au-delà
    - « *Do the simplest thing that could possibly work* »
    - « *You Aren't Gonna Need It* » (YAGNI)
  - Implémenter le strict nécessaire
    - Complètement et correctement
    - Tests unitaires compris

Conception simple (cont.)

Simplicité ≠ Facilité

- Le plus simple n'est pas forcément le plus facile
- Par exemple, dupliquer le code d'une méthode pour avoir une version légèrement modifiée est facile...
  - Méthode ayant des anomalies → Rechercher toutes les variantes pour les corriger
- Toutefois, ne pas oublier de progresser

Règles de simplicité XP

- Tous les tests doivent être exécutés avec succès
- Chaque idée doit être exprimée clairement et isolément
- Tout doit être écrit une fois et une seule
- Le nombre de classes, de méthodes et de lignes de code doit être minimal

Remaniement

Définition

« Procédé consistant à modifier un logiciel de telle façon que, sans altérer son comportement vu de l'extérieur, on en ait amélioré la structure interne »

Transformations de code

- Martin Fowler en a répertorié plusieurs dizaines
- Décrites avec grande précision
  - Présentation proche des « Design Patterns »
  - Beaucoup peuvent être appliqués mécaniquement
  - Certains IDE commencent même à en automatiser certains (i.e. Eclipse)

Exemples de résultats

- Elimination du code dupliqué
- Séparation des idées
- Elimination de code mort

Remaniement (cont.)

Lien avec les tests

- Chaque modification apportée par une transformation est minime
- Grâce aux tests on peut être certain qu'aucune erreur n'a été introduite
- On peut facilement revenir en arrière
- Comme dans la nature :
  - Chaque remaniement est une mutation isolée
  - Les tests font office de sélection naturelle
  - Les mutations nocives sont éliminées
  - La robustesse du code est accrue

Métaphores

Dans un projet classique

- On cherche à communiquer
  - Sur ce que l'on attend du projet
  - Sur ce que l'on a réalisé
- D'où un effort de rédaction non négligeable
- But : obtenir une vision commune
- Souvent noyée dans une documentation trop détaillée

XP se focalise sur la vue d'ensemble

- Conserver uniquement le strict nécessaire
- Eviter la redondance
  - En général quelques pages suffisent
  - Utilisation de métaphores
    - Eviter le jargon technique
    - Utiliser des « images »

Programmation en binôme

Fonctionnement du binôme

- Toujours deux développeurs devant une machine
- Pilote : Ecriture du code, manipulation des outils...
- Co-Pilote : Relecture continue du code, propositions...
- Dialogue permanent pour réaliser la tâche en cours

Formation des binômes

- Changement fréquent des binômes (plusieurs fois par jour)
- Sélection libre :
  - On demande l'aide de quelqu'un d'autre
  - Il ne peut refuser ou seulement temporairement
- Exemples de critères de choix :
  - (Faire) Profiter de l'expérience
  - Intérêt pour la tâche à réaliser

Programmation en binôme (cont.)

Répartition des tâches

- Chaque développeur est responsable de ses tâches
- Alors, à tout instant :
  - Il travaille sur une de ses tâches
  - Il aide quelqu'un d'autre à réaliser sa tâche
- NB : Une tâche pourra être réalisée en plusieurs fois

Une pratique « rentable » ?

- Un binôme est plus rapide sur une tâche donnée qu'un programmeur seul
- Amélioration de la qualité du code → Réduction du coût de maintenance
- Partage des connaissances, Cohésion d'équipe

Programmation en binôme (cont.)

Précautions

- Rester clair :
  - Collaboration active et continue
  - Pas « un qui code l'autre qui sur veille »
- Qualités humaines de l'équipe déterminantes
- S'assurer que le co-pilote ne « décroche » pas
- Gérer les différences de niveau (pas de marginalisation)
- S'ouvrir sur le reste du groupe en cas de problème
- Rester réaliste...
  - Il est difficile de fonctionner en binôme 100% du temps

Espace de travail

- Eviter le cloisonnement pour favoriser les échanges
- Concept de « War Room »

Responsabilité collective du code

Modèles actuels : Responsabilité individuelle

- Chaque développeur évolue dans une partie réservée
  - Nécessite de se mettre d'accord sur les interfaces
  - Limite les interférences
  - Garantie une certaine autonomie
- Notion de « responsable » souvent entâchée de « faute »
- Séparation des connaissances
  - Duplication de code
  - Pas de progression dans les compétences

Modèles actuels : Absence de responsabilité

- Code appartenant à toute l'équipe et à personne à la fois
- Tout le monde peut modifier l'application selon les besoins
  - Développement opportuniste
  - Design « plat de spaghettis »

Responsabilité collective du code (cont.)

Modèle XP

- Chaque binôme peut inter venir sur n'importe quelle partie
- Mais chacun est responsable de l'ensemble
- Par ex. un binôme peut revoir une partie peu claire du code
- Fonctionne bien uniquement dans un cadre XP
  - Tests unitaires
  - Intégration continue
  - Remaniement
  - Règles de codage

La fin des spécialistes ? → Non !

- Ils doivent devenir polyvalent
- Mais agissent comme consultant interne
  - Mise en commun des connaissances
  - Interlocuteurs privilégiés dans leur domaine

Règles de codage

Pourquoi ?

- Homogénéisation nécessaire (responsabilité collective)
  - Prise en main plus rapide du code écrit par d'autres
  - A définir avant le démarrage du projet
- Parfois mal perçues, mais
  - Adaptation assez rapide
  - Prise de conscience du travail en équipe
- Peut s'inscrire dans une démarche qualité

Aspects couverts

- Présentation du code (indentation, espaces...)
- Organisation des commentaires
- Règles de nommage (classes, méthodes, constantes...)
- Système de noms (vocabulaire commun, métaphore...)
- Idiomes de codage (parcours de liste, singletons...)

Intégration continue

Pourquoi éviter des périodes d'intégration ?

- Profiter immédiatement des efforts de chacun
- Coût de l'intégration augmente très vite avec l'éloignement dans le temps des intégrations successives
  - Fréquence d'intégration "classique" : 1/semaine
  - Fréquence XP : 1/jour à n/heure

Méthode de travail : Gestion de versions

- Test Unitaires = Tests de non-régression
- Processus suivi :
  1. Récupération d'une copie locale
  2. Modification de la copie
  3. Mise à jour de la copie locale lorsqu'elle passe les tests
  4. Correction des éventuels problèmes
  5. Mise en dépôt de la copie locale **lorsqu'elle passe les tests**

Client sur site

Notion d'équipe

- Client **intégré** à l'équipe de développement
- Equipe
- Ensemble des développeurs
- « Whole Team » (client et programmeurs)

Avantages

- Spécifications fonctionnelles généralement floues, ici :
- Client disponible pour apporter ses compétences métier
- Maitrise d'ouvrage conservée par le client
- Définition des tests de recette par le client

Une pratique inaccessible ?

- Très difficile de trouver quelqu'un de suffisamment disponible

Rythme durable

Maintenir un niveau optimal

- Savoir travailler... mais aussi se reposer !
- Beaucoup d'énergie nécessaire pour :
  - Trouver des solutions de conception simples
  - Ecrire du code propre
  - Penser à des tests unitaires
  - Explorer avec son binôme les problèmes rencontrés
- On peut faire des heures supplémentaires
  - Sur une cour te période de temps
  - Suivie d'une période de relâchement

Traiter les problèmes réels

- Règle : « *Pas d'heures sup' deux semaines de suite* »
  - Ne pas la respecter est le signe d'un problème plus profond
  - Plutôt le résoudre que le masquer par un sur plus de travail

Livraisons fréquentes

Rythment le projet

- Livraisons régulières comme point de synchronisation
- Le client fixe ces dates, par exemple :
  - Pour un projet de six ou sept mois
  - On pourra espacer les livraisons d'environ un mois et demi
  - La première livraison arrivant au bout de deux mois
- La première livraison doit arriver le plus tôt possible
  - Dissiper d'éventuels malentendus
  - Donner consistance au projet
- Les livraisons doivent être aussi proches que possible
  - Pilotage précis du projet
  - Donner des preuves de son avancement

Livraisons fréquentes (cont.)

Un « feedback » pour le client

- Mieux cerner ses besoins
- Etre rassuré sur l'avancement réel du projet

Un « feedback » pour l'équipe

- Sentiment régulier de « travail fini »
- Confrontation du produit à un environnement réel

Planification itérative

Rythme

- Environ 2 ou 3 itérations entre deux livraisons
- Pour un projet de six ou sept mois
  - Itérations de deux semaines environ

Séances de planification (« Planning Game »)

- Livraisons et itérations sont des cycles imbriqués
- Deux niveaux de granularité pour le pilotage
  - Livraisons → Fonctionnalités
  - Itérations → Tâches à réaliser par les développeurs
- Ces deux cycles sont découpés en trois phases
  1. Exploration, identifier le travail et estimer son coût
  2. Engagement, sélectionner le travail pour le cycle en cours
  3. Pilotage, contrôler la réalisation de ce qui est demandé

Planification itérative : Livraisons

Exploration (de plusieurs jours à quelques heures)

- Définir les scénarios client
  - Granularité fine (plus fine que les Use Cases)
  - Doit être testable
  - Généralement notés sur des fiches A5
    - Plus simples à manipuler
- Estimer les scénarios client
  - Attribution d'un nombre de « points » (jours théoriques)
  - Tenir compte du codage des tests et de la validation
  - En cas d'inconnue technique → prototypage **rapide**
  - Scinder/Fusionner des scénarios si nécessaire

Planification itérative : Livraisons (cont.)

Engagement (quelques heures seulement)

- Trier les scénarios (9 tas)
  - Par priorité (client) : Indispensable, Essentiel, Utile
  - Par risque (programmeurs) : Fort, Moyen, Faible
- Annoncer la « vélocité » de l'équipe
  - Nombre de points que peut traiter l'équipe en une itération
  - Pour la première itération le coach fourni sa propre estimation
- Répartir les scénarios parmi les livraisons à venir
  - Le client « achète » les scénarios en fonction de la vélocité
  - Création du plan de livraison *provisoire*

Planification itérative : Livraisons (cont.)

Pilotage (jusqu'à la date de livraison)

- Suivre les tests de recette
  - Nombre de tests réalisés par le client et les testeurs
  - Nombre de tests validés
- Gérer les défauts → Scénarios supplémentaires
  1. Intégrer au mécanisme général de planification
  2. Priorité absolue

Et ensuite ?

- Livrer sans délai → Reporter les scénarios manquants
- Célébrer l'événement et prendre du recul
  - Repas hors du lieu de travail par exemple
  - Bonne occasion pour se détendre
- Démarrage d'un nouveau cycle
  - Amélioration de la fiabilité des estimations

Planification itérative : Itérations

Exploration

- Analyser les scénarios pour les scinder en tâches
- Activité de conception (même superficielle)
- Questions au client et discussion en sa présence
- Tâche : réalisable par un binôme en une ou deux journées

Engagement

- Choisir et estimer des tâches
- Equilibrage des choix dans l'équipe
- Fusion/Scission de tâches (si nécessaire)
- Avancement/Report de scénarios (si nécessaire)
- Production du plan d'itération

Planification itérative : Itérations (cont.)

Pilotage

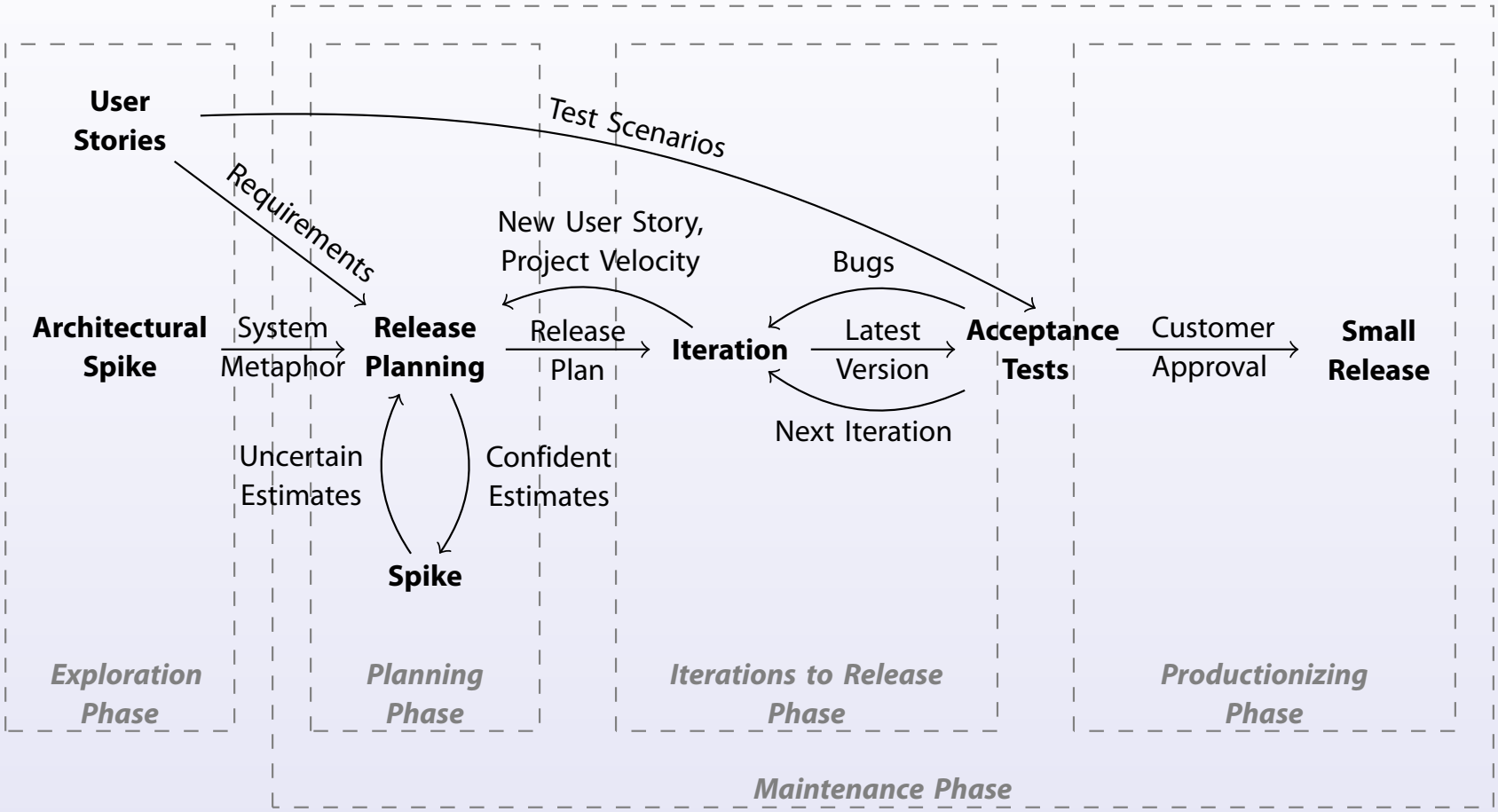
- Réalisation des tâches (mise à jour du plan d'itération)
- Tournée du Tracker (au moins deux fois par semaine)
  - Déceler tout dérapage au plus tôt
  - Prendre des mesures correctives (alléger une charge...)
- Stand-up meetings
  - Chaque matin un point sur la situation ou les difficultés
  - Pas une réunion technique ni un suivi des tâches
  - Permet d'organiser la journée (binômes...)

Et ensuite ?

- Petite livraison informelle au client de l'équipe
- Mise à jour des coûts des scénarios réalisés
- Célébrer l'événement...
  - Un pot pour repartir sur de bonnes bases

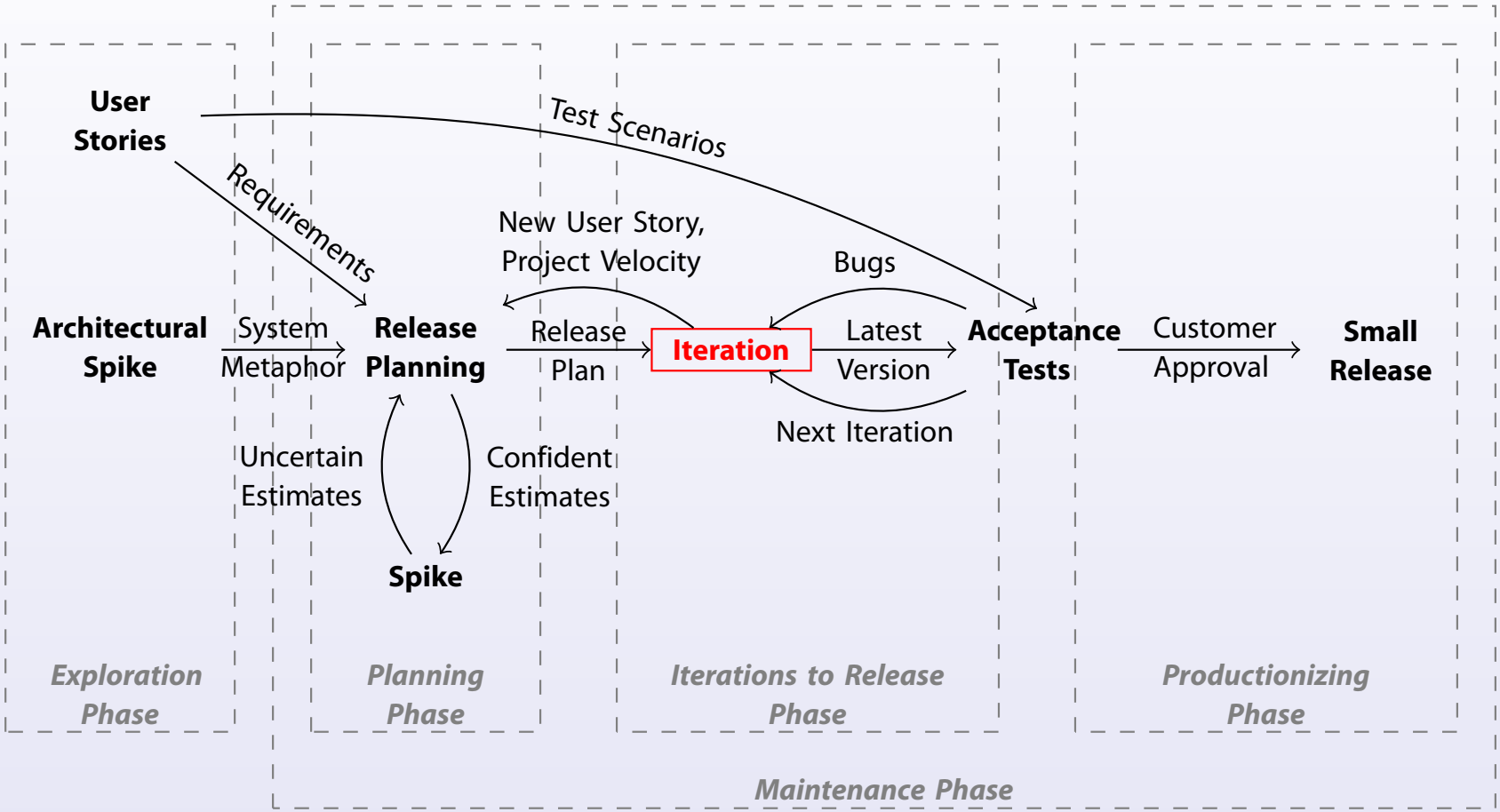
Sommaire

Cycle de vie XP



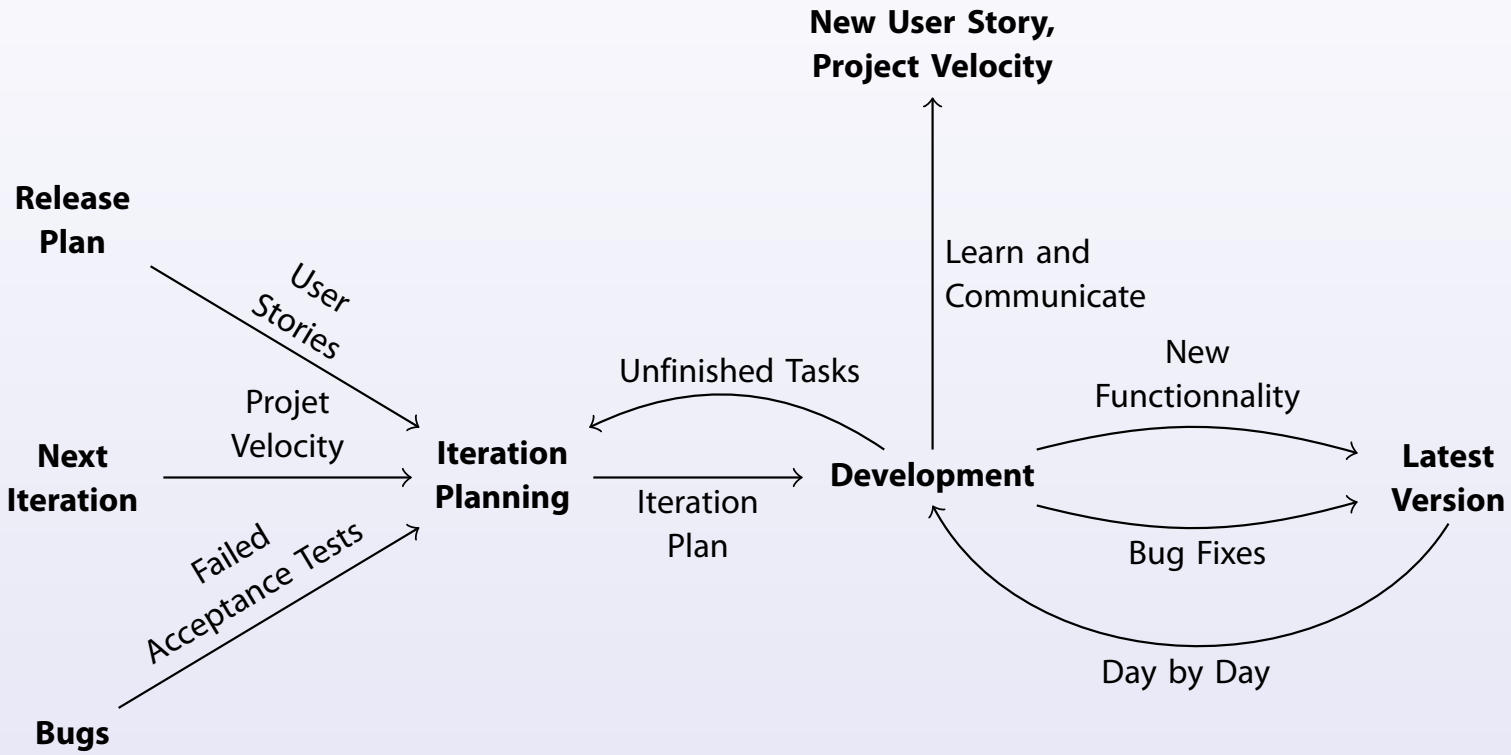
http://www.extremeprogramming.org/

Cycle de vie XP

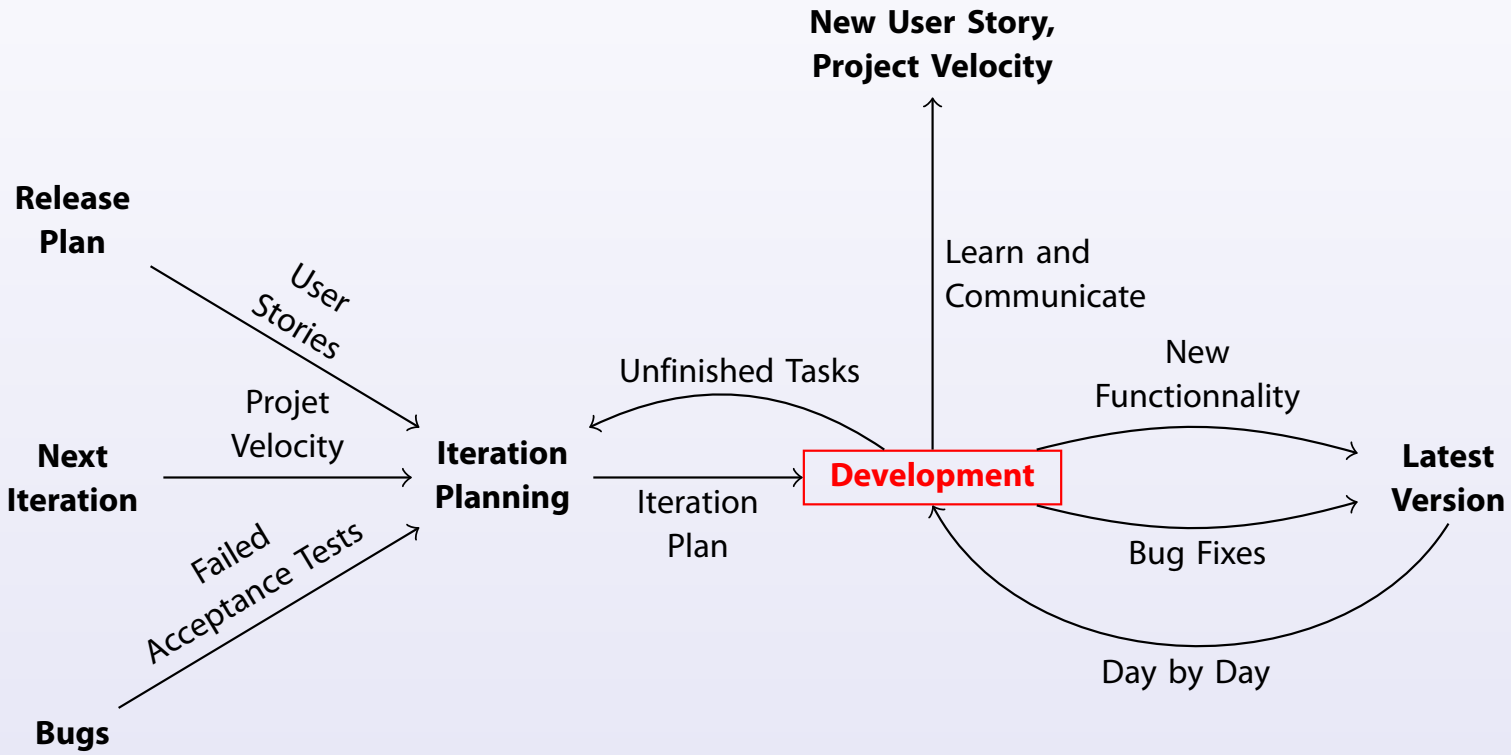


http://www.extremeprogramming.org/

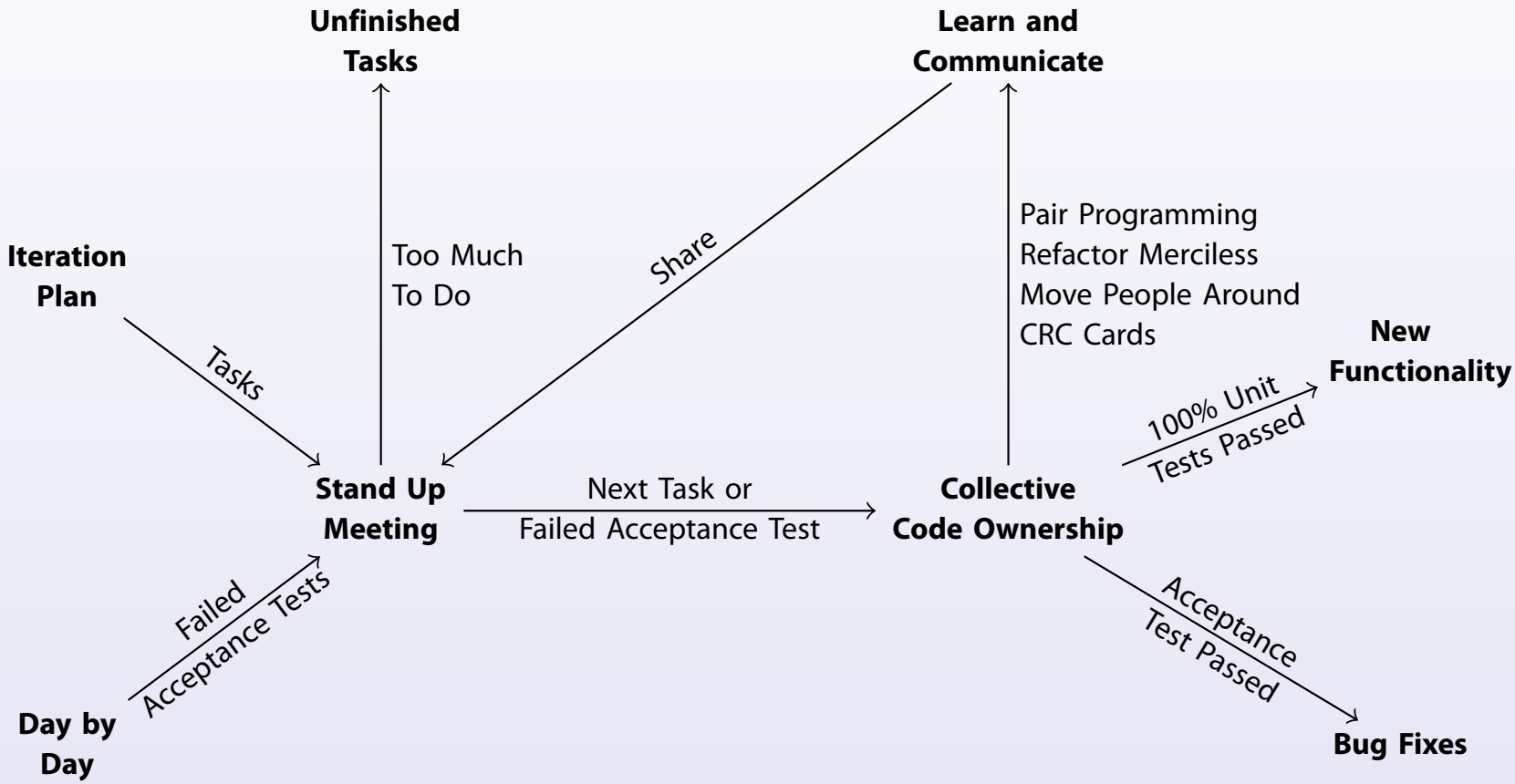
Itération


New User Story,
Project Velocity
Release
Plan
User
Stories
Learn and
Communicate
Unfinished Tasks
New
Functionnality
Projet
Velocity
Next
Iteration
Iteration
Planning
Development
Latest
Version
Iteration
Plan
Failed
Acceptance Tests
Bug Fixes
Day by Day
Bugs

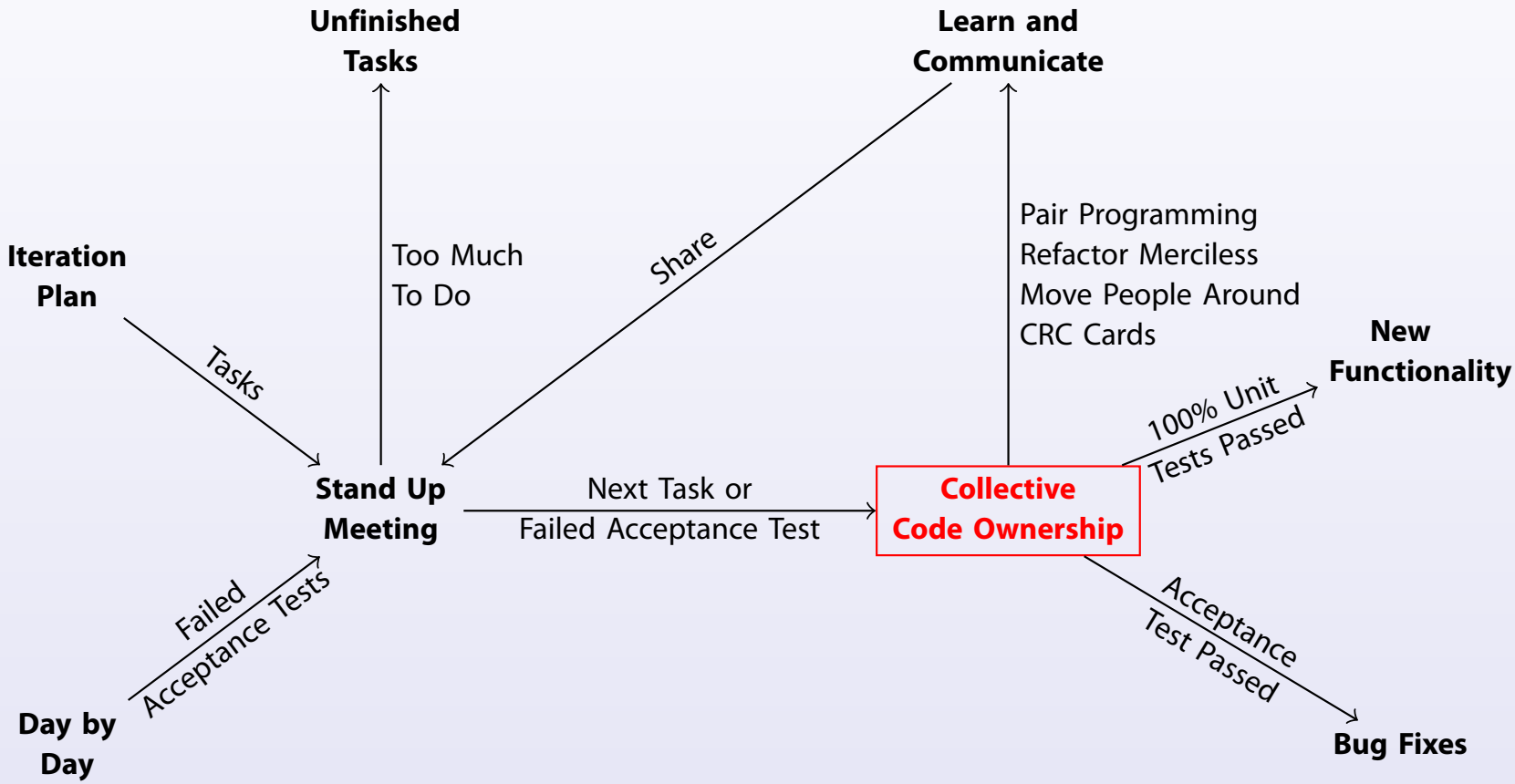
Itération


New User Story,
Project Velocity
Release
Plan
User
Stories
Learn and
Communicate
Unfinished Tasks
New
Functionnality
Projet
Velocity
Next
Iteration
Iteration
Planning
Development
Latest
Version
Iteration
Plan
Bug Fixes
Failed
Acceptance Tests
Day by Day
Bugs

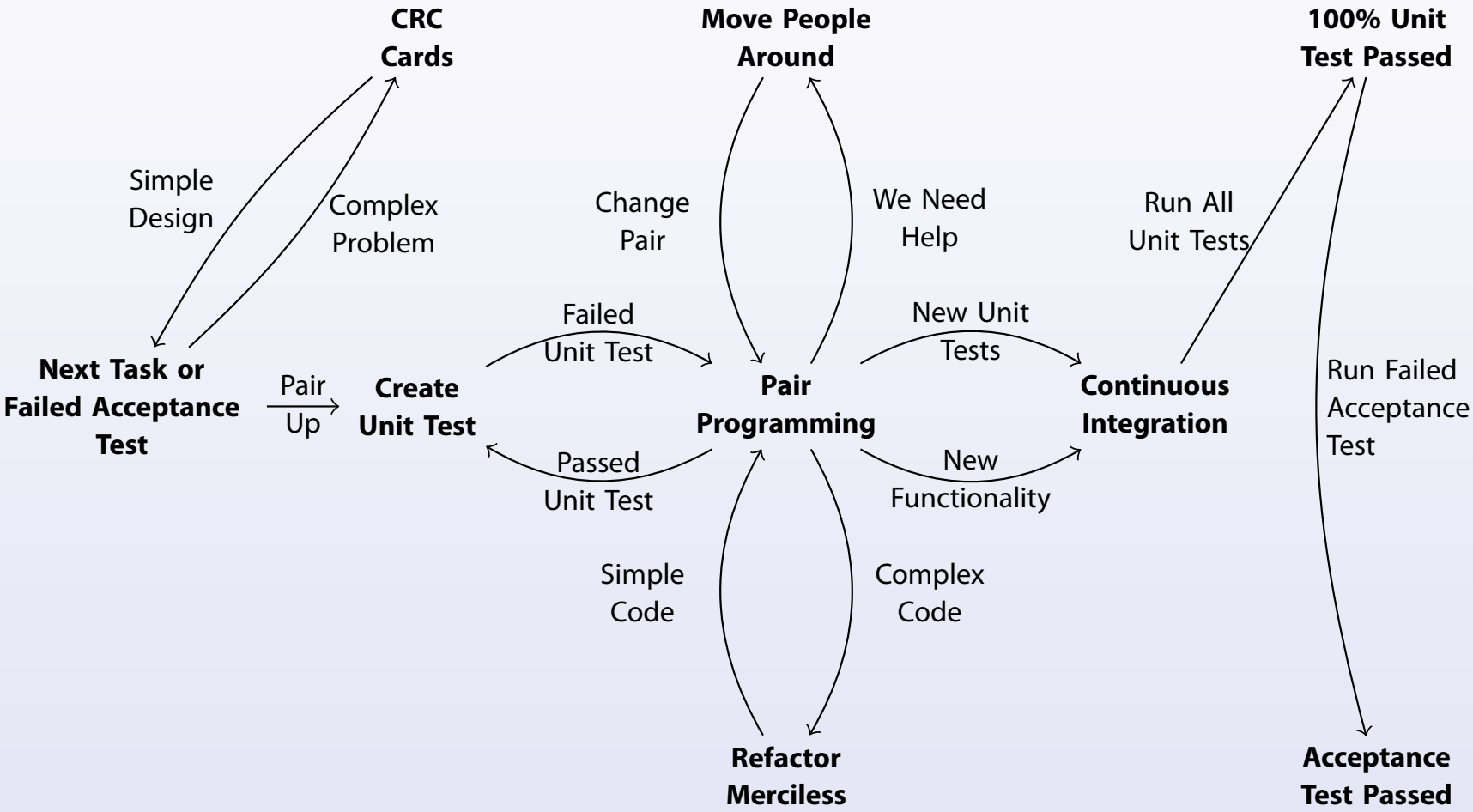
Développement



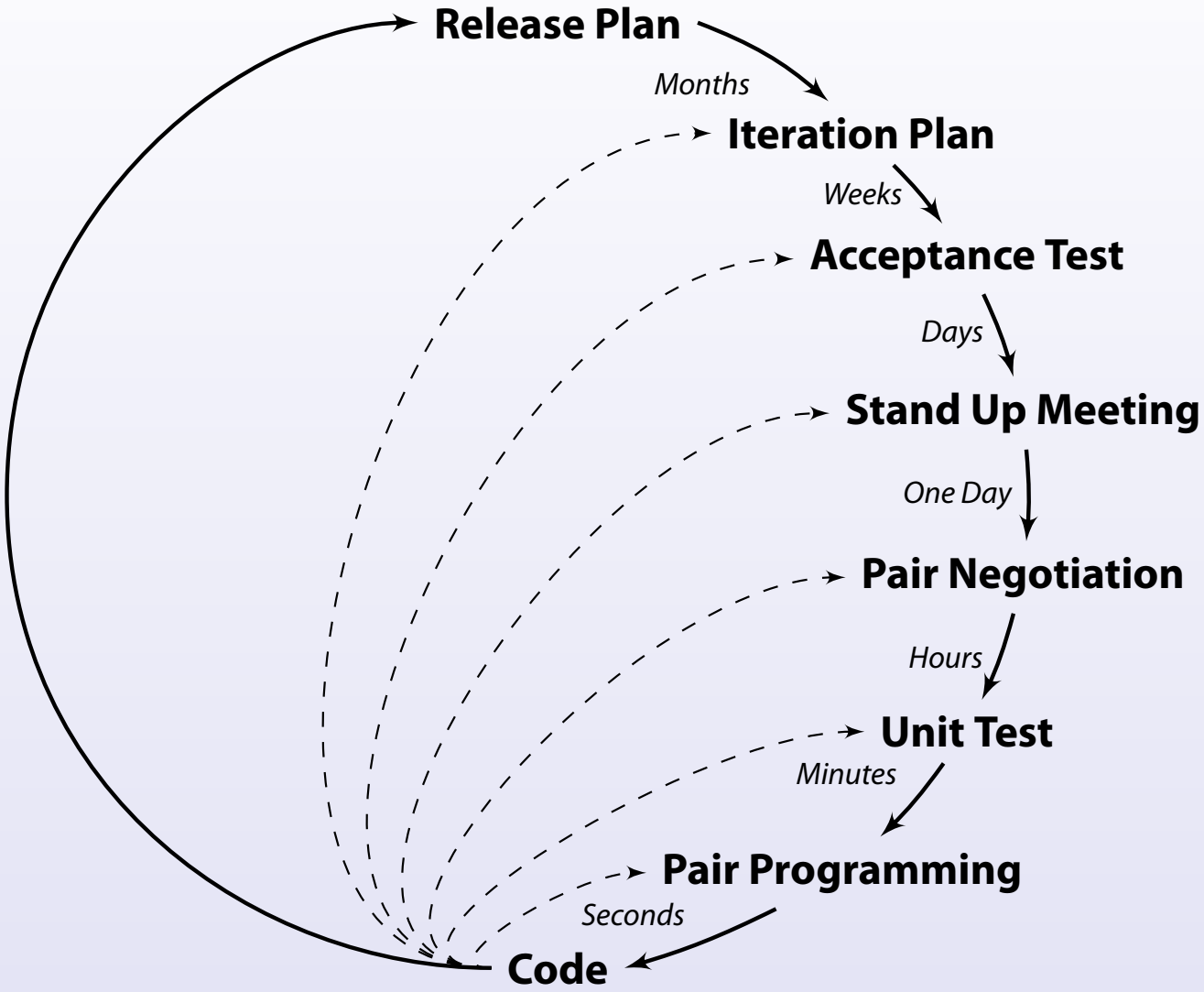
Développement



Le code appartient à tous



Feedback


Release Plan
Months
Iteration Plan
Weeks
Acceptance Test
Days
Stand Up Meeting
One Day
Pair Negotiation
Hours
Unit Test
Minutes
Pair Programming
Seconds
Code

Sommaire

XP et la modélisation

Can you use UML with XP ?

Yes. You can apply the artifacts of the UML – activity diagrams, class diagrams, collaboration diagrams, component diagrams, deployment diagrams, sequence diagrams, statechart diagrams, and use case diagrams – when you are taking an XP approach to development.

How do you use UML with XP ?

Minimally you should apply Agile Modelling and use UML artifacts on your XP project only where appropriate. Ideally you should apply all of the principles and practices of Agile Modelling when doing so.

XP et la documentation

Au centre des débats

- « *On n'écrit pas de documentation dans XP* » : Faux !
- XP tend à éliminer la paperasse... mais restons nuancés

Documentation utilisateur

- Manuel utilisateur, Procédure d'installation...
- Ils doivent être écrits, mais planifiés comme les tâches techniques

Documents liés au projet

- Comptes rendus, Plannings...
- Le client étant présent dans l'équipe, l'importance de ces documents est **réduite**

XP et la documentation (cont.)

Documentation technique

- Document de conception, Diagrammes...
- Tests fonctionnels → Spécification
- Tests unitaires → Conception détaillée
- Remaniement → Modification aisée de la conception
- Le code est de la documentation (génération automatique ?)

Autour du programmeur

- Code source comme principale production du projet
- Programmeur comme acteur essentiel
- Contrairement à certaines autres méthodes
  - Il n'est pas un simple exécutant
  - Il dispose de pratiques pour être plus efficace
  - Il accède à une certaine reconnaissance

Principaux facteurs de succès

Sur le plan Humain

- Présence d'un coach potentiel
- Quelques développeurs expérimentés... et ouverts
- Une équipe capable de travailler... en équipe

Sur le plan Organisationnel

- Environnement de travail adapté (« War Room »)
- Hiérarchie consentante
- Culture d'entreprise adaptée
  - Pas de mérite basé sur les heures supplémentaires
  - Pas de jeu politique (gagnant-gagnant)
  - Pas d'attachement aux méthodes linéaires, ou aux tonnes de documents comme reflet de la qualité

Principaux facteurs de succès (cont.)

Sur le plan Technique

- Possibilité de travailler sur des portions réduites de l'application
  - Valable dans le cas d'une reprise d'un gros projet
  - Redécouper le logiciel dans un tel cas
- Langage de programmation permettant des mécanismes d'abstraction et de factorisation
  - La plupart des langages objets entrent dans cette catégorie
- Disposer d'un outil de gestion de versions efficace

Le problème du contrat...

Aujourd'hui en France...

- Mise en œuvre d'XP encore peu présente en entreprise
- Parfois pour des développements internes

Contrats forfaitaires

- Un fournisseur s'engage à un résultat donné, dans un délai donné pour un prix fixé
- Limites du modèle
  - Périmètre fonctionnel rarement clair

  → Rapports conflictuels

  → Tout le monde perd
- Fige un maximum de paramètres (cahier des charges...)
  - Difficile de pratiquer XP dans ces conditions
  - Sauf pour deux entreprises ayant une relation partenariale
  - Impose l'utilisation d'avenants au contrat

Le problème du contrat... (cont.)

Contrats d'assistance technique (« régie »)

- Un fournisseur met à disposition du personnel compétent
  - Le client assure maîtrise d'ouvrage et maîtrise d'œuvre
  - Le fournisseur facture au client le temps passé
- Limites du modèle
  - Repousse simplement la question de la maîtrise d'œuvre
  - Quelle est l'expérience du client dans ce domaine ?
  - Motivation des collaborateurs « mis à disposition » peut être un problème
- La possibilité d'appliquer XP dépendra donc du client
  - S'il est capable de le faire en interne
  - Sinon une intervention extérieure peut permettre la transition vers XP

Le problème du contrat... (cont.)

Contrats d'assistance forfaitée

- Combinaison des deux modèles
  - Facturation au temps passé
  - Mais le fournisseur reste maître d'œuvre
- Généralement difficile à faire accepter au client
- Nécessite une relation de confiance entre les deux parties

Difficultés spécifiques à la France

- Culture institutionnelle française
  - XP est une méthode privilégiant les réalités du terrain
  - En France, les décisions partent du management pour se répercuter sur les opérationnels
- Directions financières attachées au contrat forfaitaire
  - Budget prévisionnel dans un périmètre bien défini
  - Besoins réels de l'utilisateur financièrement inexistants

Avoir l'œil critique

Critiques possibles à l'encontre d'XP

- Trop anarchique
- Pas assez de documentation
- Trop extrême...
- Culte du super programmeur

Une utopie ?

- Méthode profondément humaniste (cf. manifeste agile)
- Tout le monde est-il honnête ?

Maîtriser XP...

... c'est savoir quand ne pas utiliser XP

... c'est savoir adapter XP aux besoins courants

Autres méthodes agiles

Methodology	Summarizing Phrase
XP	Simplicity
Scrum	Prioritized Business Value
Lean	Return on Investment (ROI)
FDD	Business Model
AUP	Manage Risk
Crystal	Size and Criticality
DSDM	Current Business Value

http://www.devx.com/architect/Article/32836

Autres méthodes agiles (cont.)

Condition	XP	Scrum	Lean	FDD	AUP	Crystal	DSDM
Small Team	✓	✓	✓	✗	✗	-	✓
Highly Volatile Requirements	✓	✓	✓	✓	-	-	✗
Distributed Teams	✗	✓	✓	✓	✓	✗	✗
High Ceremony Culture	✗	✗	-	-	✓	-	✓
High Criticality Systems	✗	-	-	-	-	✓	✗
Multiple Customers / Stakeholders	✗	✓	✓	-	-	-	✗

The table illustrates which conditions favor (✓), discourage (✗), or are neutral (-)

http://www.devx.com/architect/Article/32836

Pour plus d'informations sur XP

- *L'eXtreme Programming*
  J.L. Bénard, L. Bossavit, R. Médina, D. Williams
  Editions Eyrolles
- *Extreme Programming Explained*
  Kent Beck, Cynthia Andres
- XProgramming.com
  http://xprogramming.com/
- Extreme Programming : A gentle introduction
  http://www.extremeprogramming.org/
- eXtreme Programming France
  http://xp-france.net/
- Design Up
  http://www.design-up.com/