



Chapitre 2: *Méthodes Agiles*



Olfa Daassi

Agilité

- ▶ L'origine du terme **Agile** remonte au Manifeste Agile (2001), publication pour laquelle dix-sept experts du domaine du développement logiciel se sont réunis afin de débattre les points communs entre leurs différentes méthodes de développement.
- ▶ Ainsi naît le Manifeste se basant sur quatre valeurs et douze principes présentés ci-dessous :



Agilité

▶ Valeurs de l'agilité

- ▶ **Les Individus et leurs interactions** plus que les processus et les outils ;
- ▶ **Des logiciels opérationnels** plus qu'une documentation exhaustive ;
- ▶ **La collaboration avec les clients** plus que la négociation contractuelle ;
- ▶ **L'adaptation au changement** plus que le suivi d'un plan



Agilité : Les Individus et leurs interactions

- ▶ Les outils et les processus sont importants, mais il est plus important que des personnes compétentes travaillent ensemble efficacement. Un exemple serait : l'entreprise veut mettre en place des protocoles pour la réalisation des interactions entre les différentes équipes, mais le protocole lui-même prend du temps à réaliser.
- ▶ Ceci arrive souvent dû au fait que l'organisation mette en place ses processus en anticipant le besoin, et pas forcément de la façon la plus efficiente. Une approche agile pour ceci pourrait être de laisser les interactions se mettre en place naturellement, puis venir formaliser dans un processus si cela s'avère nécessaire ;



Agilité : Des logiciels opérationnels

- ▶ Une bonne documentation est utile pour aider les gens à comprendre comment le logiciel est construit et comment l'utiliser, mais le principal point de développement est de créer un logiciel, pas une documentation.
- ▶ L'organisation des demandes d'évolution par les utilisateurs en *User Stories* est la matérialisation de cette valeur, une fois que ça représente une documentation suffisante pour que le développeur puisse travailler sur le logiciel ;



Agilité : La collaboration avec les clients

- ▶ Un contrat est important mais ne remplace pas une collaboration étroite avec les clients pour découvrir ce dont ils ont besoin. Dans les autres méthodologies de gestion de projet informatique, le travail de développement ne commence qu'après une longue étape de négociation du contrat. Le client doit spécifier toutes ses demandes avant que tout développement soit fait.



Agilité : La collaboration avec les clients

- ▶ Dans une implémentation agile, le travail de développement peut commencer sans que tout soit forcément défini, et avoir le client en tant que collaborateur permet à ce que celui-ci fasse évoluer les demandes par rapport à l'évolution de sa propre compréhension du sujet. Bien sûr qu'il subsiste un contrat au sens commercial du terme avec une approche agile. Dans cette troisième valeur du Manifeste Agile, la notion de contrat concerne plus la façon de travailler au quotidien avec le client. Il est important de mettre en place une relation de travail avec les différentes parties prenantes du client plus collaborative qu'une relation client/fournisseur;
-

Agilité : L'adaptation au changement

- ▶ Un plan de projet est important, mais il ne doit pas être trop rigide pour tenir compte des changements technologiques ou environnementaux, des priorités des parties prenantes et de la compréhension des gens du problème et de sa solution. Ce n'est pas pour dire que dans l'agile il n'y a pas de plan, mais que ceux si sont adaptés régulièrement, à chaque cycle de développement, afin d'apporter le maximum de valeur par rapport aux besoins du client. Les courtes périodes de temps dans lesquelles s'organisent le développement permet que ces changements puissent être rapidement pris en compte, et qu'à la fin le logiciel soit plus en accord avec l'évolution du besoin du client.
-



Méthodes AGILE

▶ **Principe :**

- ▶ Le terme "Agile" définit une approche de gestion de projet qui prend en quelque sorte le contre-pied des approches traditionnelles prédictives et séquentielles (cycle en V ou waterfall).
- ▶ En effet, une approche dite traditionnelle attend généralement du client une expression détaillée et validée du besoin en entrée de réalisation, laissant peu de place au changement.



Méthodes AGILE

► **Effet :**

- La réalisation dure le temps qu'il faut et le rendez vous est repris avec le client pour la recette.
- Cet **effet tunnel** peut être très néfaste et conflictuel, on constate souvent un déphasage entre le besoin initial et l'application réalisée
- Certains projets se terminent dans la douleur (surtout dans le cadre d'un projet signé au forfait) au risque de compromettre la relation client
- Certaines fonctionnalités demandées se révèlent finalement inutiles à l'usage alors que d'autres, découvertes en cours de route, auraient pu donner plus de valeur au produit



Méthodes AGILE

- ▶ **Effet :**
- ▶ Parmi les motifs d'échecs, arrivent en tête :
 - ▶ Manque d'implication des utilisateurs finaux : 12,8 %
 - ▶ Spécifications incomplètes : 12,3 %
 - ▶ Changement de spécifications en cours de projet : 11,8 %

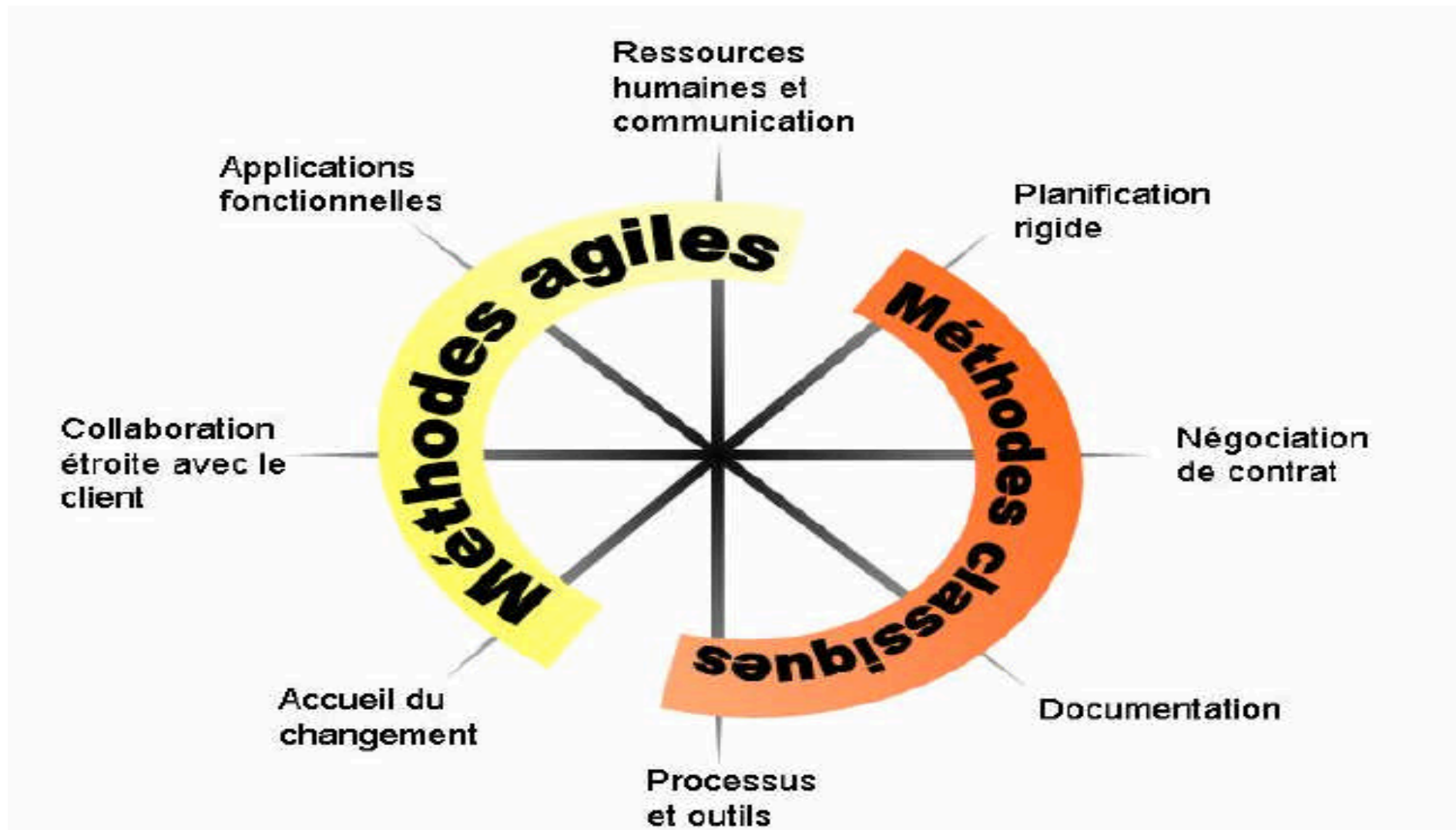
Méthodes AGILE

▶ **Bases :**

- ▶ L'approche Agile propose au contraire de supprimer purement et simplement cet effet tunnel
 - ▶ en donnant davantage de visibilité,
 - ▶ en impliquant le client du début à la fin du projet et
 - ▶ en adoptant un mode itératif et incrémental. Elle considère que le besoin ne peut être figé et propose au contraire de s'adapter aux changements. Mais pas sans un minimum de règles.



Méthodes AGILE



Méthodes AGILE

▶ **Exemples:**

- ▶ Une approche Agile part du principe que spécifier et planifier dans les détails l'intégralité d'un produit avant de le développer (approche prédictive) est contre productif.
 - ▶ Cela revient à planifier dans les détails un trajet "Tunis - Djerba" en voiture par les petites routes.,
 - ▶ Spécifiant chaque ville et village traversée, l'heure de passage associée, chaque rue empruntée dans les agglomérations, litres d'essence consommés, kilomètres parcourus, etc.



Méthodes AGILE

▶ **Exemples:**

- ▶ Les imprévus ne manquerons pas d'arriver : embouteillages, déviations, travaux, sens de circulation inversés, voir la panne, etc. Rendant votre planification et vos spécifications très vite obsolètes.
- ▶ Combien de temps aurez vous passé à planifier cet itinéraire,
- ▶ Comment réagirez vous face à vos frustrations de ne pas pouvoir appliquer votre plan à la lettre ?



Méthodes AGILE

▶ **Exemples:**

- ▶ Une approche Agile consiste à se fixer un premier objectif à courts termes (une grande ville par exemple) et se lancer sur la route sans tarder
- ▶ Une fois ce premier objectif atteint, on marque une courte pose et on adapte son itinéraire en fonction de la situation du moment
- ▶ Et ainsi de suite jusqu'à atteindre la destination finale (approche empirique)



Méthodes AGILE

- ▶ Dans le cadre d'un projet informatique Agile, le client élabore sa vision du produit à réaliser et liste les fonctionnalités/exigences de ce dernier.
- ▶ Il soumet cette liste à l'équipe de développement qui estime le coût de chacune d'entre elles, on peut ainsi se faire une idée approximative du budget global.



Méthodes AGILE

- ▶ L'équipe sélectionne ensuite une portion des exigences à réaliser dans une portion de temps courte appelée itération.
- ▶ Chaque itération inclut des travaux de conception, de spécification fonctionnelle et technique quand c'est nécessaire, de développement et tests.
- ▶ A la fin de chacune de ces itérations, le produit partiel est montré au client.
- ▶ Ce dernier peut alors se rendre compte par lui même très tôt du travail réalisé, de l'alignement sur le besoin



Méthodes AGILE


- ▶ L'utilisateur final quant à lui peut se projeter dans l'usage du produit et émettre des feedbacks précieux pour les futures itérations
- ▶ A la fin de chacune de ces itérations, le produit partiel est montré au client.
- ▶ Les risques quant à eux sont levés très tôt



Méthodes AGILE : panorama

- ▶ **eXtreme Programming**
- ▶ Dynamic Software Development Method
- ▶ Adaptive Software Development
- ▶ Crystal Clear
- ▶ **SCRUM**
- ▶ Feature Driven development
- ▶ **[UP ?]**





Méthodes Avancées : eXtreme Programming

Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team changes. The team members change.

The problem isn't change, because change is going to happen; the problem, rather, is our inability to cope with change.

eXtreme Programming

Pères de la méthode : Ward Cunningham et Kent Beck
(1996). 5 valeurs clés :



► Communication

- rendre la communication omniprésente entre tous les intervenants

► Simplicité

- il coûte moins cher d'aller au plus simple et de rajouter des fonctionnalités par la suite plutôt que de concevoir dès le départ un système très compliqué dont on risque de n'avoir jamais l'utilité



eXtreme Programming

- ▶ **Feedback**

- ▶ indispensable pour que le projet puisse accueillir le changement

- ▶ **Courage**

- ▶ concerne aussi bien les développeurs que le client

- ▶ **Respect**

- ▶ respecter et être respecté en tant que personne



eXtreme Programming

- ▶ Feedback rapide
- ▶ Assumer la simplicité
- ▶ Changements incrémentaux
- ▶ Accueillir le changement à bras ouverts
- ▶ Un travail de qualité
- ▶ Apprendre à apprendre
- ▶ Faible investissement au départ
- ▶ Jouer pour gagner (comme au foot jouer pour gagner et non pour éviter de perdre)

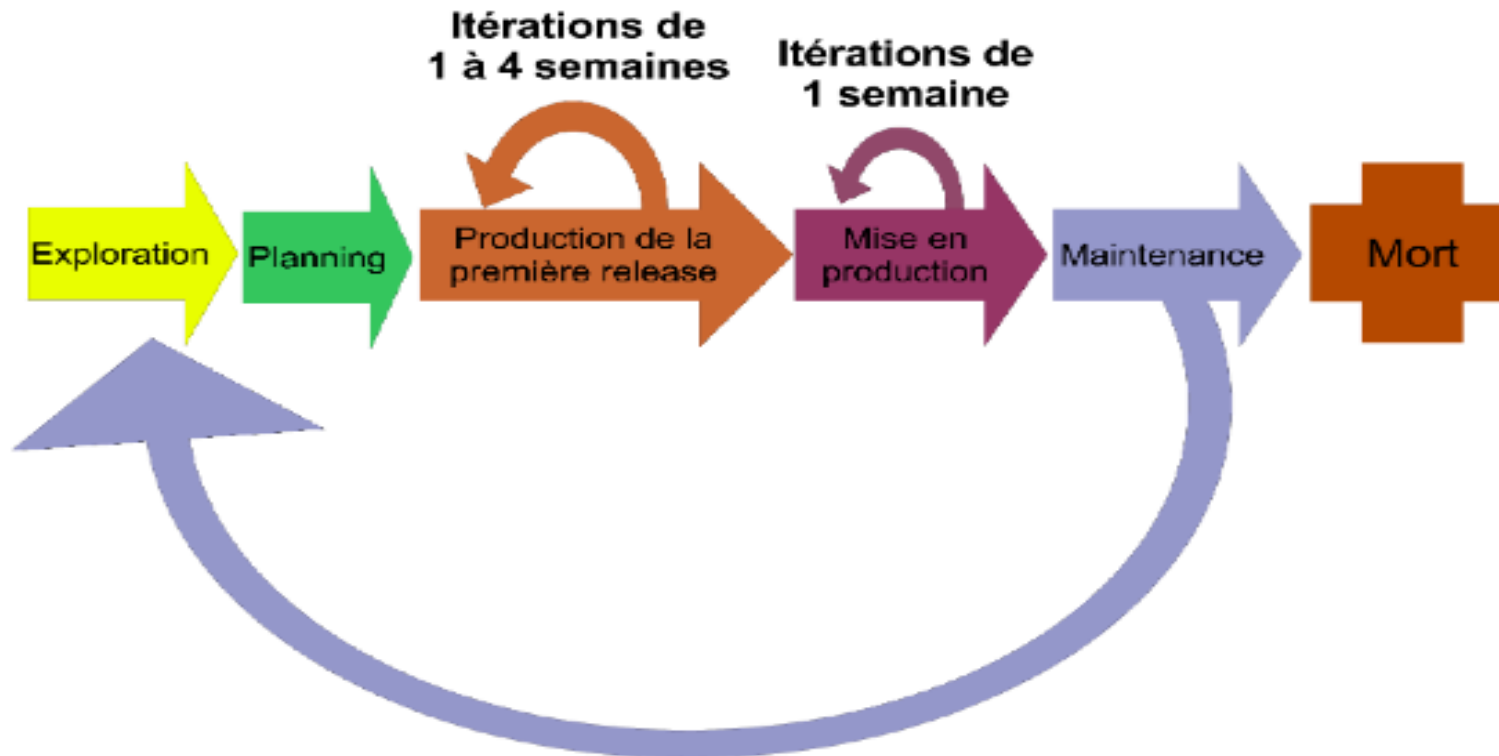


eXtreme Programming

- ▶ Des expériences concrètes : une séance de conception doit donner des résultats efficaces
- ▶ Communication ouverte et honnête : dire si on a besoin d'aide, dire que le code est incomplet ...
- ▶ Responsabilités acceptées : permettre à chacun de prendre des responsabilités
- ▶ Adaptation aux conditions locales : pas forcément appliquer XP à la lettre
- ▶ **Voyager léger : utiliser le juste nécessaire**
- ▶ Mesures honnêtes : par exemple dire “cela prendra plus ou moins deux semaine que dire cela prendra 14 176 heures”



eXtreme Programming



Maintenance = ajouter de nouvelles fonctionnalités

eXtreme Programming

- ▶ Exploration : explorer les différentes possibilités d'architecture pour le système
 - ▶ Le client exprime ses besoins en termes de fonctionnalités sous forme de user stories
 - ▶ Par exemple pour un logiciel de gestion de carnet d'adresses on pourrait écrire les scénarios suivants :
 - ▶ “Je rentre un nom ou un prénom, et le logiciel affiche la liste de toutes les personnes qui possèdent ce nom ou ce prénom”
 - ▶ “Je peux enregistrer mon carnet d'adresses au format HTML”



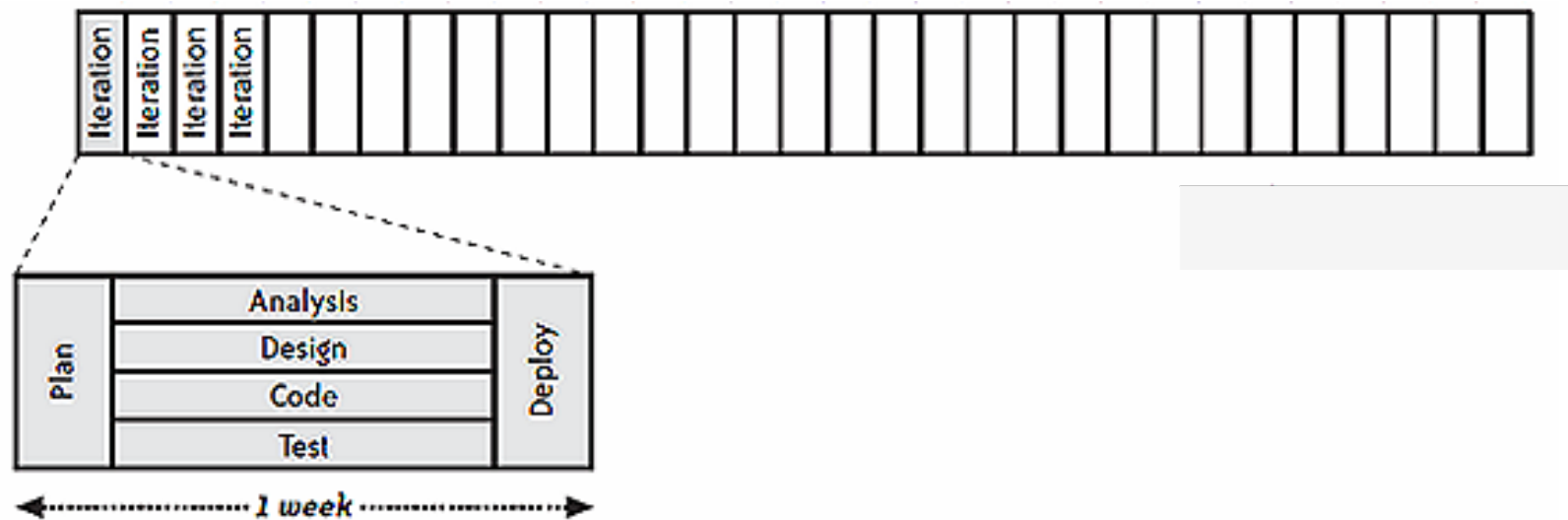
eXtreme Programming

- ▶ Planning : classement des user stories par ordre de priorité
- ▶ Itérations jusqu'à première release : c'est le développement proprement dit de la première version de l'application et ça se fait sous forme d'itérations de 1 à 4 semaines



eXtreme Programming

► Itération



eXtreme Programming : Principes

- ▶ Mise en production : des tests sont menés au cours de cette phase et les développeurs procèdent à des réglages affinés pour améliorer les performances
- ▶ Maintenance : adjoindre les fonctionnalités secondaires qui avaient volontairement été laissées de côté jusque là
- ▶ Mort : la fin du projet intervient quand le client n'arrive plus à écrire des user stories supplémentaires
 - ▶ Ceci signifie que tous ses besoins ont été satisfaits



eXtreme Programming : Principes

- ▶ Refactoring du code : retravailler le code pour le rendre plus lisible et plus robuste
- ▶ Programmation en 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
 - ▶ Changement fréquent des binômes (plusieurs fois par jour)
 - ▶ (on sélectionne librement l'aide de quelqu'un pour profiter de l'expérience par exemple)
 - ▶ Avantages : Un binôme est plus rapide sur une tâche donnée qu'un programmeur seul, amélioration de la qualité du code, partage des connaissances

eXtreme Programming : Principes

- ▶ Rester clair :
 - ▶ Collaboration active et continue
 - ▶ Pas “un qui code l’autre qui surveille”
 - ▶ S’assurer que le Co-Pilote ne “décroche pas”
 - ▶ Gérer les différences de niveau
 - ▶ S’ouvrir sur le reste du groupe en cas de problème



eXtreme Programming : Principes

- ▶ Propriété collective du code
- ▶ Intégration continue (plusieurs fois par jour)
- ▶ Pas de surcharge de travail : ne pas dépasser 40 heures de travail par semaine
- ▶ Client sur site : présence sur site d'une personne minimum à temps plein pendant toute la durée du projet
- ▶ Standards de code (normes de nommage et de programmation)



Rôles dans XP

- ▶ **Développeur (extreme programmer)**
 - ▶ travaille en binôme, communique
 - ▶ doit être autonome
 - ▶ a une double compétence : développeur – concepteur
 - ▶ estime les stories, définit les tâches d'ingénierie, estime le temps que vont prendre les stories et les tâches, implémente les stories et les tests unitaires



Rôles dans XP

▶ Client

- ▶ doit apprendre à exprimer ses besoins sous forme de user stories
- ▶ a à la fois le profil de l'utilisateur et une vision plus élevée sur le problème et l'environnement du business
- ▶ doit apprendre à écrire les cas de tests fonctionnels
- ▶ il a l'autorité de trancher sur les questions concernant les stories



Rôles dans XP

▶ Testeur

- ▶ a pour rôle d'aider le client à choisir et à écrire ses tests fonctionnels
- ▶ affiche les résultats sur des graphiques
- ▶ fait en sorte que les gens sachent quand les résultats des tests déclinent

▶ Tracker

- ▶ aide l'équipe à mieux estimer le temps nécessaire à l'implémentation de chaque user story
- ▶ contrôle la conformité de l'avancement au planning



Rôles dans XP

▶ Coach

- ▶ recadre le projet
- ▶ ajuste les procédures
- ▶ planifie des réunions
- ▶ fait en sorte que le processus de la réunion soit suivi
- ▶ note les résultats de la réunion pour un rapport futur et les passe au tracker
- ▶ Il ne dit pas aux gens :
 - ▶ ce qu'ils doivent faire (le Client et le plan d'iteration le font)
 - ▶ quand ils doivent finir (planning d'engagement le défini)
- ▶ Il ne vérifie pas comment les gens s'en sortent



Rôles dans XP

- ▶ **Consultant**

- ▶ n'apporte pas de solution toute faite
- ▶ apporte à l'équipe les connaissances nécessaires pour qu'elle résolve elle-même les problèmes

- ▶ **Big Boss**

- ▶ apporte à l'équipe courage et confiance

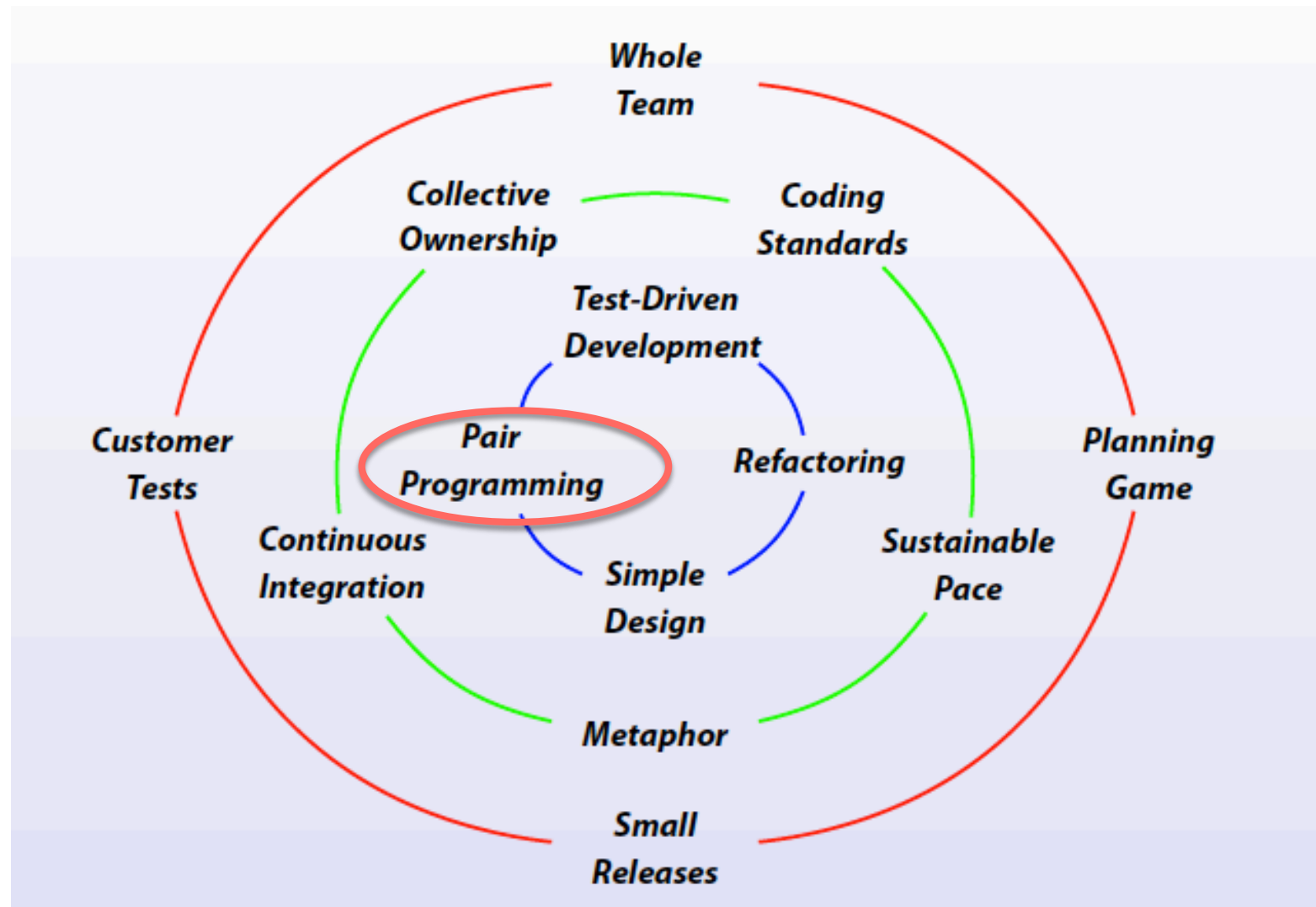


Rôles dans XP

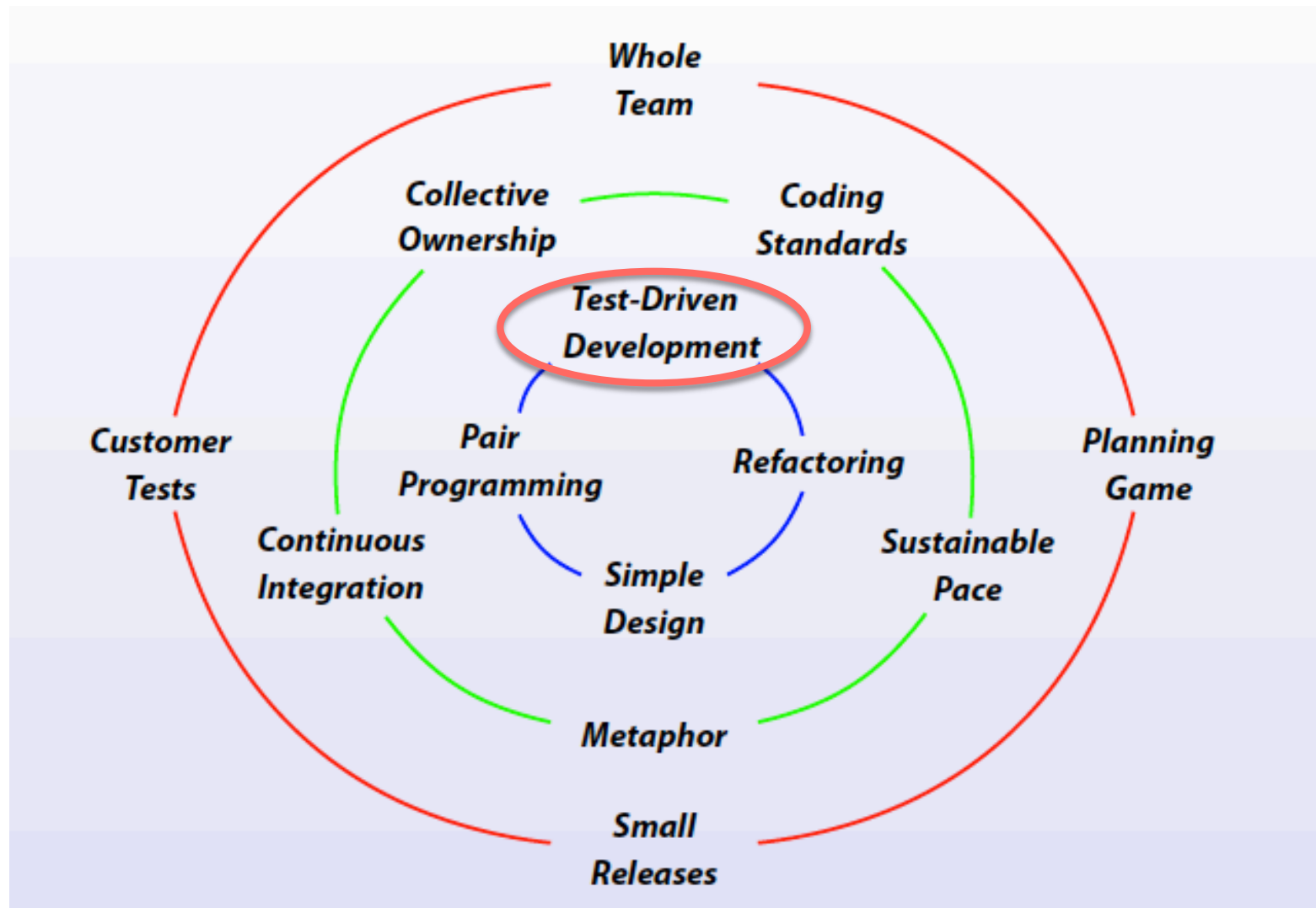
- ▶ Certains rôles peuvent être combinés par une même personne
- ▶ Par exemple : une même personne peut avoir les rôles de Coach et de Tracker
- ▶ Certains rôles ne devraient pas être combinés :
 - ▶ Développeur-Tracker, Développeur-Testeur, Client-Développeur, Coach-Testeur
 - ▶ Le Coach ne peut être combiné avec d'autres à part le Tracker



Pratiques de l'XP



Pratiques de l'XP



Test-Driven-Devlopement

- ▶ Développement piloté par les tests
- ▶ En complément des tests de recette qui servent à prouver au client que le logiciel remplit ses objectifs, XP utilise intensivement les Test Unitaires de non-regression
 - ▶ Ces tests sont écrits par les développeurs en même temps que le code lui-même pour spécifier et valider le comportement de chaque portion de code ajoutée
- ▶ Typiquement, un test unitaire porte sur une classe ou une fonction précise.
 - ▶ Il simule un contexte d'exécution, déclenche un traitement et vérifie le résultat de ce traitement



Test-Driven-Devlopement

- ▶ **Automatisation :**
 - ▶ Tests d'un logiciel sont généralement automatisables
 - ▶ De nombreux langages possèdent leurs canevas de tests unitaires (JUnit pour Java, SUnit pour SmallTalk, PerlUnit ...)
 - ▶ L'automatisation fait moins de temps consacré aux tests et ce dû à l'automatisation
 - ▶ Temps nécessaire à l'automatisation long mais :
 - ▶ Moins de défauts dans le code
 - ▶ Moins de régressions

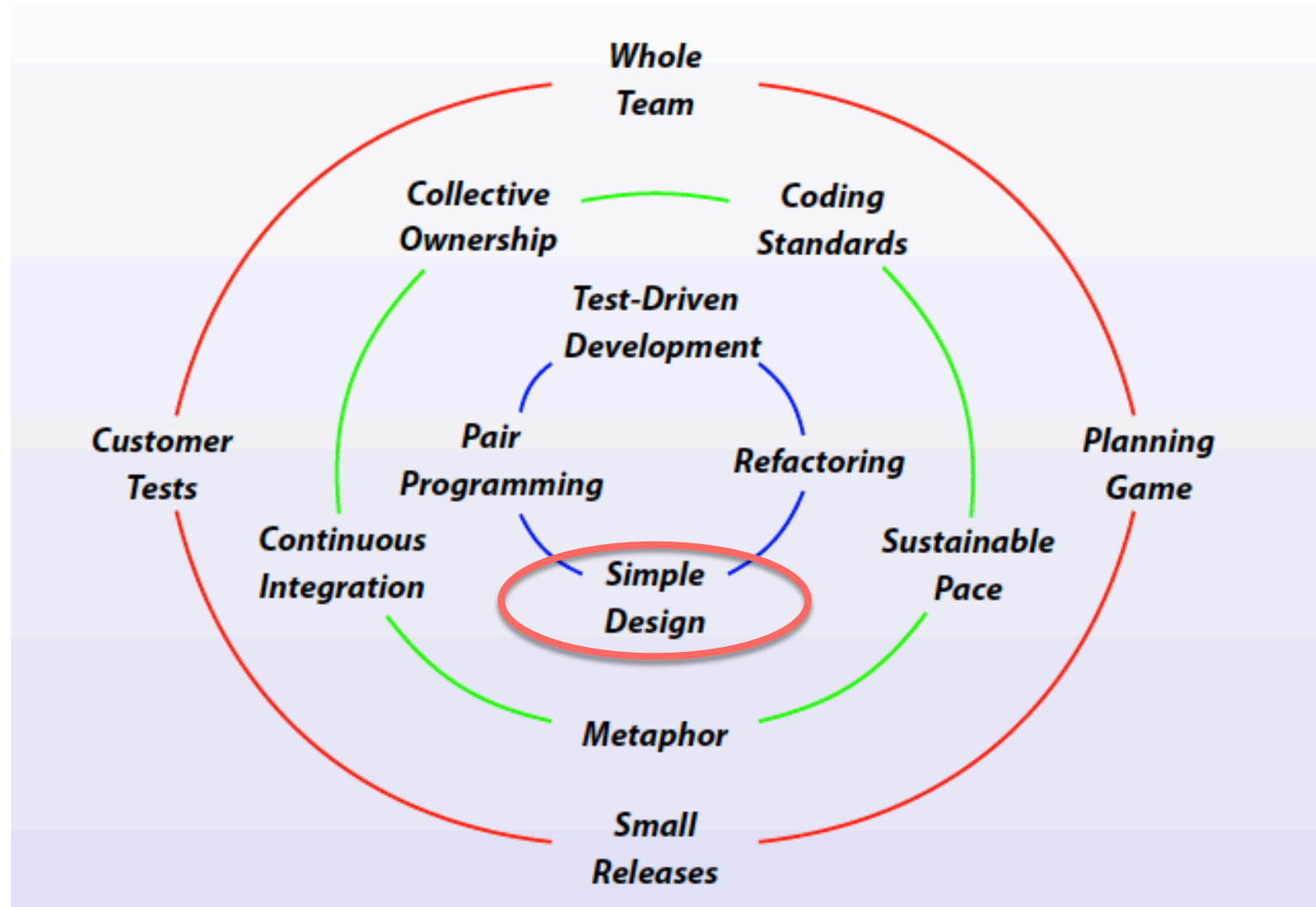


Test-Driven-Devlopement

- ▶ **Ecrire les tests en premier**
 - ▶ Position à priori paradoxale, mais ..
 - ▶ Elimination du dilemme habituel en fin de projet
 - ▶ Code plus facilement testable
 - ▶ Meilleure conception
 - ▶ Tests unitaires ➔ forme de conception détaillée



Pratiques de l'XP



Conception Simple

- ▶ Conception = Investissement (en temps et/ou complexité)
- ▶ Dans XP : se concentrer sur une et une seule fonctionnalité
 - ▶ Obtenir la meilleure conception possible
 - ▶ “Do the simplest thing that could possibly work”
- ▶ Implémenter le strict nécessaire
 - ▶ Complètement et correctement
 - ▶ Tests unitaires compris

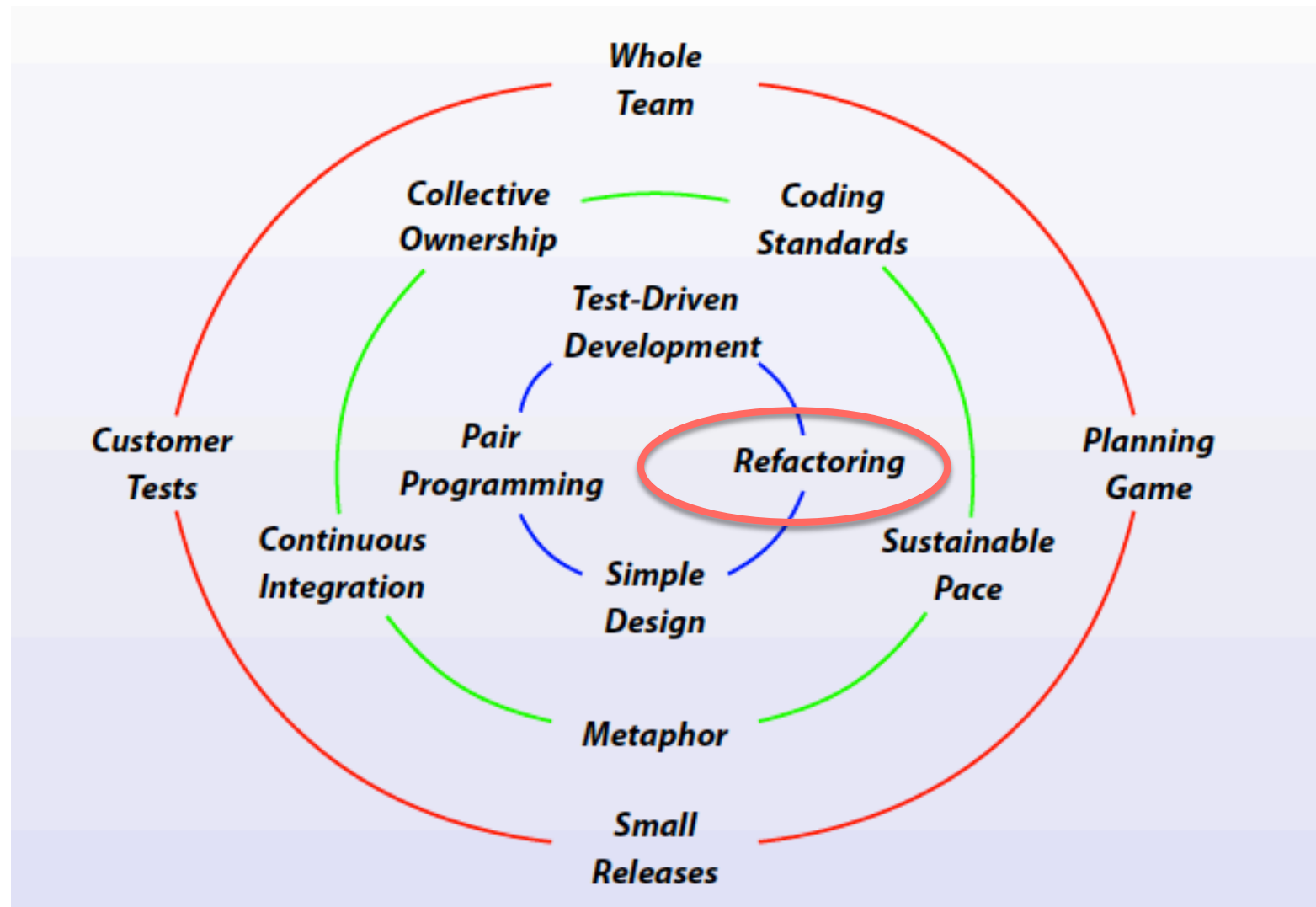


Conception Simple

- ▶ Simplicité \neq facilité
- ▶ Le plus simple n'est pas forcément le plus facile
- ▶ Règles de simplicité dans 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



Pratiques de l'XP



Refactoring (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”

- ▶ **Exemples de résultats du remaniement :**

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



Refactoring (Remaniement)

- ▶ Il s'agit donc de :
 - ▶ Retoucher ou réorganiser des portions de code existantes a comportement fonctionnel identiques en vue d'améliorer la structure d'ensemble de l'application
- ▶ L'une des motivations principales de remaniement est l'absence de duplication dans le code : tout doit y apparaître “Once and Only Once”
- ▶ Lorsqu'un développeur doit utiliser une partie de code qui se trouve déjà quelque part, il remanie l'application de manière à pouvoir utiliser directement le code existant
 - ▶ Ainsi, les modifications de l'application n'impactent généralement que peu d'endroits dans le code



Refactoring (Remaniement)

- ▶ Le remaniement est un facteur d'assurance de simplicité :
 - ▶ Il s'agit de supprimer au fur et à mesure tout ce qui nuit à la simplicité et peut ralentir l'équipe
- ▶ Comme les tests unitaires, le remaniement est pratiqué à la longueur de journée :
 - ▶ Il s'agit d'une discipline quotidienne de qualité et non d'une activité épisodique de réécriture



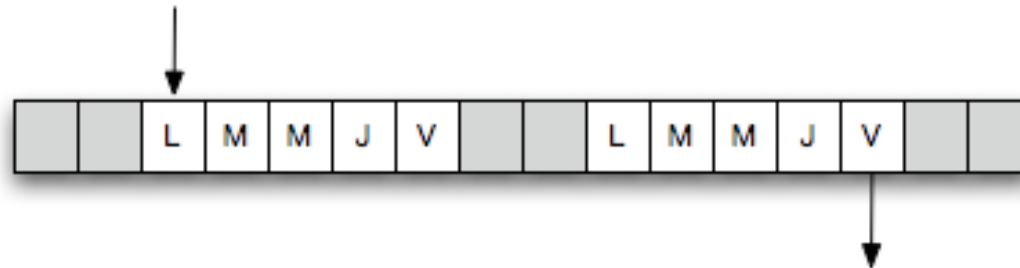
Retour sur le Cycle XP

- ▶ Dans un projet XP, la durée du cycle de développement correspond à la durée d'une itération c'est à dire typiquement deux semaines
- ▶ Remarque : durée du cycle de développement = temps qui s'écoule entre le moment où l'on décide d'implémenter une fonctionnalité et celui où l'on met en production une nouvelle version du logiciel qui intègre la fonctionnalité en question



Retour sur le Cycle XP

- ▶ Une itération dans XP a la structure suivante :



- ▶ Le premier jour de l'itération est consacré à la réunion de **planification**
 - ▶ Y a un planning de livraison et un planning d'itération
 - ▶ Planning de **livraison** : Il s'agit d'une réunion pour énoncer l'ensemble du projet. Ce planning est ensuite utilisé pour construire les plannings d'itérations

Retour sur le Cycle XP : Planification

- ▶ **Planning d'itération** : Une réunion de prévision d'itération se déroule au début de chaque itération pour produire le planning des tâches de développement de cette itération
- ▶ Les User Stories (tâches) sont choisies pour cette itération par le client depuis le planning de livraison par ordre de rentabilité
- ▶ Le client doit choisir des user stories dont le total des estimations atteint **la Vélocité** de l'itération précédente



Retour sur le Cycle XP : Planification

- ▶ La vélocité : L'équipe donne une estimation de sa vélocité c'est à dire le nombre de points de scénarios qu'elle s'estime capable de traiter en une itération (par exemple 10 points)
- ▶ Au tout début du projet, cette vélocité est seulement estimée, mais après les premières itérations elle est réajustée en adoptant la règle suivante :
 - ▶ La vélocité estimée pour une itération donnée correspond exactement au nombre de points effectivement traités à l'itération précédente.
- ▶ Remarque : La vélocité ne représente en aucun cas le niveau de performance de l'équipe. Cependant, ses variations peuvent indiquer des problèmes passagers qu'il ne faut pas ignorer



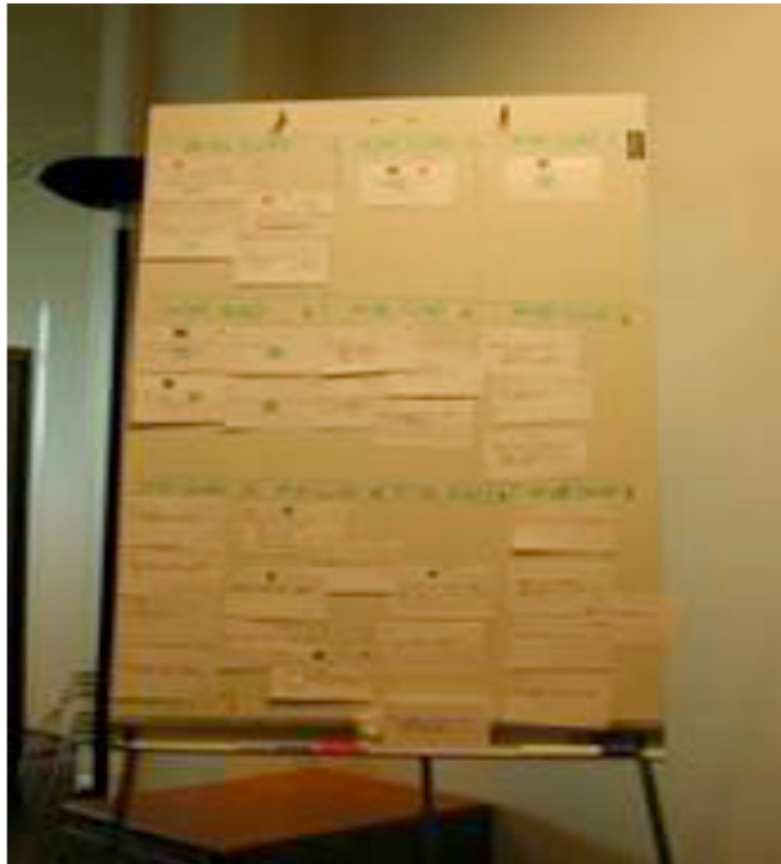
Retour sur le Cycle XP

- ▶ Le client établit lui-même le plan de développement en affectant les différents scénarios aux itérations à venir, de sorte qu'à chaque itération la somme du nombre de points des scénarios choisis soit égale à la vitesse annoncée



Retour sur le Cycle XP

- Pour ce faire, l'équipe peut se doter d'un outil basique qui consiste à noter les scénarios sur des fiches cartonnées que l'on colle sur un grand tableau ou un mur





Retour sur le Cycle XP

- ▶ Suite à cette phase de planification, l'équipe s'organise pour réaliser les tâches en question:
 - ▶ Elle prend en charge le suivi des tâches ainsi que les activités d'analyse du besoin, de conception, d'implémentation et de test correspondantes.
 - ▶ Il est important de noter qu'il n'y a pas de changement de cap intermédiaire : l'équipe se focalise sur son objectif sans interruption jusqu'à la fin de l'itération



Retour sur le Cycle XP

- ▶ Au terme des deux semaines de l'itération, l'équipe met une nouvelle version du logiciel à disposition du client
 - ▶ Ce logiciel est robuste, testé et sa structure interne est laissée aussi propre que possible



Give the Team a Dedicated Open Working Space



Give the Team a Dedicated Open Working Space

- ▶ Communication is very important to an Extreme Programming (XP) team. You can add vital communication paths to your team by just taking down the barriers that divide people. Putting computers in a central area that no one owns makes pair programming easier and encourages people to work together with feelings of equal value and contributions. Putting a few desks around the perimeter gives people a place to work alone without becoming disconnected from the rest of the team.



Give the Team a Dedicated Open Working Space

- ▶ Adding white boards for design sketches and important notes or blank walls where user story cards can be taped adds even more channels for communication.



Référence

- ▶ <http://www.extremeprogramming.org/>

