Intégration des Technologies

Samuel "Big Boss" Monroe

30 Mai 2015

Table des matières

Chapitre 1

Avant-propos

Toi, mauvais élève qui n'a jamais été aux cours de Mr. Thiry, avais sans doute comme seul choix d'étudier des slides un peu austères pour l'examen.

En bonne âme, je viens à toi avec cette synthèse d'une pureté ultime afin de te sauver de ton funeste destin.
Composée de ce que j'ai retenu, complété via l'aides des internets, et corrélées avec les informations d'autres membres de la F0X.

En espérant que tu sois membre de la F0X en mettant les mains sur ce document.

Chapitre 2

Historique

1 Le Chaos - 1960

L'historique des méthodes organisationnelle présenté ici naît avec l'apparition de l'informatique mais plus spécifiquement des langages de programmation, qui signifient que l'on va pouvoir mettre sur pied des projets de développement logiciels pour couvrir les nouveaux besoins d'applications de l'entreprise.

Dans les alentours de 1960 on pourra citer le **FORTRAN** en 1954, le **COBOL** en 1959 et enfin le **BASIC** en 1964.

Tout ceci étant relativement nouveau, chaque entreprise s'organise selon sa propre conception sur base des **échéances**, du **budget** ou encore de la **taille du projet**.

La période de Chaos des gestions de projets est comme son nom peut le faire comprendre, une période d'**échec** sur cette gestion.

2 Waterfall - 1970

Pour pallier à cette période de Chaos un pionier du développement logiciel et scientifique informatique Américain, **Winston W. Royce**, va mettre au point le modèle **Waterfall** pour définir une manière de manager le développement de larges logiciels.

Sa méthode est calquée sur ce qui se fait dans l'industrie en Amérique, qui met en avant la **planification** du projet et la maximisation de l'**éfficacité** dans chacun des maillons d'une chaîne de production.

Ce modèle est nommé **Waterfall** car représente une cascade où vont se succéder :

— **Requirements** : On documente les besoins et les spécifications du produit.
— **Design** : Modélisation de l'architecture du logiciel
— **Implémentation** : Le développement en lui-même
— **Vérification** : Tests, vérification du bon fonctionnement
— **Maintenance**

Cette méthode va se révéler un **échec**, entraînant un système de bureaucratie contre-productif et une perte de motivation chez les acteurs du projet.

3 Lightweight - 1990

Les méthodologies légères vont conduire à l'émergence de méthodologies plus spécifiques apparentées que l'on développera juste après.

Lightweight apprenant des erreurs de Waterfall va proposer une nouvelle façon d'organiser cette gestion de projets :

— **Pas de prédictions** : La planification à l'extrême chez Waterfall n'a pas fonctionné.
— **Adaptif** : Les changements étant inévitables, et ayant justement conduit à arrêter de tout prévoir, il faut à ce qu'il y aie des changements, les accepter et les gérer éfficacement.
— **Centré sur les gens** : Waterfall ayant cloisonné les rôles et fait perde la motivation aux acteurs, on va se recentrer sur les gens et travailler avec eux plutôt que de se concentrer sur le process.
— **Pas de documentation exagérée**, conduisant à la bureaucratie, à des marges de manoeuvres limitées et probablement à la perte de motivation.

3.1 Crystal - 1992

Cette famille de méthodologies à été développée par **Allistair Cockburn** dans le milieu des années 90.

Les méthodologies Crystal sont centrées sur les gens, leur interaction, la communauté, les compétences et talents et la communication.

Ces méthodologies sont faciles d'adoption.

3.2 DSDM - 1994

Dynamic **S**oftware **D**evelopment **M**ethod, consortium créé par des vendeurs et experts, est une compilation de bonnes pratiques dans la gestion de projets.

DSDM s'appuie sur 9 principes de bases :

1. Implication des utilisateurs dans le cycle de développement
2. Autonomie
3. Visibilité du résultat
4. Adéquation
5. Développement itératif
6. Réversibilité
7. Synthèse
8. Tests
9. Coopération

3.3 SCRUM - 1995

Métaphore liée à la mêlée du rugby utilisée pour la première fois dans une publication Japonaise de Hirotaka Takeuchi et Ikujiro Nonaka dans le monde industriel.

Cette publication va inspirer **Ken Schwaber** et **Jeff Sutherland** dans ce que deviendra la méthode SCRUM.

3.4 Rational Unified Process (RUP) - 1996

Donne un cadre au développement logiciel avec un grand nombre de variantes pas seulement agiles.

Fournit un ensemble d'outils décrivant les processus et pratiques à mettre en place pour gérer un projet.

3.5 Extreme Programming - 1999

Inventée par **Kent Beck**, **Cunningham** et **Jeffries** pendant un travail chez Chrysler.

Méthode agile qui pousse à l'extrême des principes simples, proposant une solution d'organisation pour des petites équipes avec des besoins qui changent fréquemment.

4 Agile - 2000

Groupe de pratiques ayant pour base le **Agile Manifesto** rédigé en Février 2001 par 17 personnes conceptrices de méthodologies Lightweight, qui vont extraire les principes et critères de leurs méthodologies qui selon eux conduisent aux meilleures gestion de projets, afin de les réunir et créer le **Agile Manifesto**.

5 Lean

Henri Ford, inspiré par le **Scientific Management**, va organiser sa production de voiture sur ce modèle en découpant les tâches dans la chaîne à l'extrême, en associant chaque petite tâche simple et répétitive à un seul ouvrier afin de maximiser sa production.

Cependant, au Japon, les Toyoda vont s'organiser différemment et forment leurs ouvriers en les rendants compétents et en leur donnant des responsabilités et qui doivent devenir polyvalents.

L'américain **W. Edwards Deming** va enseigner aux japonais comment améliorer la qualité, conception, tests et ventes de leurs produits.
Taiichi Ohno quant à lui va développer la méthode Just-in-Time et celle des **five 0** chez Toyota.

L'industrie Japonaise va rattraper cele Américaine, dont l'éfficacité des ouvriers était pourtant neuf fois supérieures, en proposant des produits moins chers et de meilleure qualité.

Les Poppendieck vont publier en 2003 le livre **Lean Sofware Development**, en se basant sur l'histoire de l'industrie qui précède.

Chapitre 3

Méthodologies Waterfall

1 Waterfall - Objectifs

Waterfall a pour objectif de résoudre les problèmes de mauvais résultats de projets qui peuvent survenir.
Cette méthodologie va donc proposer l'organisation sur un **planning** et d'éviter le plus possible de s'en écarter, ainsi que de proposer le contrôle de ce qui est produit.
Cela va avoir pour inconvéniants :

— Bureaucratie
— Retards
— Démotivation
— Des besoins clients non remplis

2 Waterfall - Modèle général

Waterfall s'établit en **cascade** avec 4 processus :

1. **Analyse** : Fait l'analyse fonctionnelle du produit à développer pour l'équipe de design
2. **Design** : Donne les spécifications techniques pour l'équipe chargée de l'implémentation
3. **Implémentation** : Implémente le projet et rend la documentation pour l'équipe de tests, effectue les changements suite aux rapports de test de l'équipe de test
4. **Tests** : Reçoit le produit et effectue les tests, rend les rapports de test à l'équipe d'implémentation

3 Waterfall - Modèle en V

On peut également établir un cycle en V à propos de Waterfall :

— Analyse fonctionnelle -> Test d'acceptance
— Design général -> Tests d'intégrations
— Design détaillé -> Tests unitaires
— Développement

4 Waterfall - Rôles

Waterfall divise la gestion des projets en trois rôles, le **Client** en relation directe avec le **Fournisseur**, supervisés par un **Commité de Pilotage**.

5 Gestion du temps via Diagrammes

Le diagramme de **Gantt** permet de visualiser dans le temps les diverses tâches composant un projet, de même que celui de **PERT** inventé en 1958 par la NAVY.

6 Modèle de la construction et développement

En construction, il faut établir les plans et le planning et puis organiser la construction autour de deux-ci, le projet suit deux phases :

— **Modélisation :** Compliquée et dont le coût est faible
— **Implémentation :** Longue, coût total est élevé

Waterfall calque son modèle sur celui de l'industrie et de la construction, hors le développement ne peut pas suivre ce modèle, les besoins sont imprévisibles et des changements peuvent survenir à tout instant du processus.

Là où un **modèle** est utile en construction, un **prototype** est plus intéressant pour du développement.

7 Waterfall - Quelques définitions

— **Gestion de programme** : Encadre les projets
— **Project Management Office** : Surveille les projets (processus et métriques)

8 Prince2

Créé en 1996 par le gouvernement UK, mis à jour en 2009, propose des cours et certifications (énorme business).

PRINCE2 (**PR**ojects **IN C**ontrolled)) est une méthode de gestion de projets structurée autour de **7 principes**, **7 processus** et **7 thèmes**.

Les 7 **Principes** :

— **Justification d'affaires en continu** : Savoir pourquoi on fait les choses
— **Retours d'expérience** : Prendre en compte les expériences passées
— **Rôles et responsabilités définis** : Définir qui fait quoi
— **Management par séquence** : Découper le travail
— **Management par exception** : Tolérer certaines marges et ne pas remonter tout à la hierarchie
— **Focalisation sur les produits** : Toujours travailler vers la réalisation du produit
— **Adaptation à l'environnement du projet** : Prendre l'environnement en compte

Les 7 **Processus** :

— **Elaborer le projet** : Nommer les gens, créer les documents à remplir, rédaction des objectifs, vérification des projets précédents, validation par le commité de pilotage
— **Diriger le projet** : Comité de pilotage : décisions, approbations
— **Initialiser le projet** : Plannification du projet en détail par le chef de projet
— **Contrôler une séquence** : Chef de projet fait le suivi du projet, préparation des tâches, suivi des rapports d'avancement, identifie tout risque ou déviation par rapport au plan initial et rapporte au comité de pilotage
— **Gérer la livraison du produit** : Les exécutants rapportent sur la progression
— **Gérer une limite de séquence** : A la fin d'une séquence, le chef de projet fournit un rapport au comité de pilotage
— **Clore le projet** : Le chef de projet s'assure que tout ele travail est achevé et documente les retours d'expérience

Les 7 **Thèmes** :

— **Cas d'affaire** : La raison du projet
— **Organisation** : Les rôles et responsabilités
— **Qualité** : Produit conforme aux exigences client
— **Planification** : Dates, quantités, détails
— **Risque** : EValuer et maîtriser les risques
— **Changement** : Catégoriser, valider, contrôler
— **Progression** : Moyens de contrôle de l'avancement

9 PMI/PMBOK

Le Project Management Institute produit en 1983 un guide définissant les champs de connaissances couvrant la gestion d'un projet : le **P**roject **M**anagement **B**ody of **Knowledge**.
Celui-ci recense les bonnes pratiques professionnelles dans la gestion de projet, et est standardisé ANSI en 1999.

Equivalant à PRINCE2.

10 CMM/CMMI

Capability **M**aturity **M**odel 1991 (+ **I**ntegration 2001).

Commandé à la Software Engineering Institute par le DoD Américain, recueil de bonnes pratiques pour évaluer et améliorer les entreprises, s'appliquait d'abord au domaine informatique, ensuite à tous les domaines d'ingénierie.

5 niveaux de maturité :

1. **Initial :** Aucun processus défini, pas de monitoring, réactions dans l'urgence
2. **Managed :** Chef de projet établit des plans de développement, assurance qualité
3. **Defined :** Centralisation des bonnes pratiques, suivies par tous
4. **Quantitatively Managed :** Les processus sont mesurés et contrôlés par rapport aux besoins du client
5. **Optimizing :** Amélioration continue et anticipation

11 ITIL

Information Technology Infrastructure Library.

Créé par le gouvernement UK dans les années 1990, ensemble d'ouvrages recensant les bonnes pratiques du management du système d'information.

Processus définis et contrôlés, traçabilité des actions.

12 Waterfall - Critique de la standardisation

Le standardisation a les avantages de proposer des bonnes pratiques pour améliorer l'éfficacité, d'éviter aux gens de réinventer la roue et de proposer un reporting uniforme permettant une gestion facilitée du projet.

Cependant celle-ci amène une lourdeur bureaucratique et une documentation exagérée, ainsi qu'un manque de flexibilité alors que celle-ci est importante dans un projet de développement.

Ces standards sont aussi bien souvent un business, les certifications étant payantes.

Cela dit, celle-ci est très importantes pour des domaines sensibles où le contrôle doit être important (centrales nucléaires, armée, ...)

13 UML

Unified Modeling Language, permet la visualisation d'un système, est normalisé et documenté, propose 14 types de diagramme.

Chapitre 4

Méthodologies LightWeight

1 Méthodologies "Légères"

Les 4 précèptes importants de ces méthodologies vont être les suivants :

— De l'organisation et pas de structure cadenassée
— Adaptation aux changements et pas de prédictions sur le futur
— Le fonctionnement est centré sur les gens, pas sur l'organisation
— Pas de documentation exagérée

2 Les besoins changent

Les changements étant **inévitables** et le mieux est de les accepter et de faire avec, on pourrait même en tirer profit.

Dans les fonctionnements lourds, le planning étant fixé une fois pour toute, des changements de besoins au niveau du client seront facturés en plus car s'écartant du contrat initial.

En lightweight, le développement est itératif et le client participe activement à celui-ci, de sorte qu'il puisse introduire des demandes de changements qui orientent la suite du développement et n'ont pas d'impact majeurs puisque ces changements sont "anticipés".
Ceci donne un avantage compétitif aux développeurs.

3 Gérer le changement

Les changements sont inévitables, il faut juste réagir rapidement et éfficacement à ceux-ci. Les itérations se font sur quelques semaines, proposent à chaque issue des fonctionnalités fonctionnelles et utiles au client, et celui-ci va pouvoir donner son avis sur ce qu'il peut déjà utiliser afin de proposer des modifications utiles si besoin qui seront implémentées dans la prochaine itération.

4 Le Client

Le client veut principalement un **prix à l'avance** et surtout ne pas **payer plus que prévu**.
En fonctionnant sur cette base d'itérations et de reviews clients, on met à jour le planning selon les nouveaux besoins et les nouvelles idées, il n'est pas nécéssaire non plus de surfacturer des changements ce qui engendre de la confiance de la part du client car ses volontés principales sont respectées.
Il faut cependant lui faire comprendre qu'un investissement en temps est nécéssaire de sa part pour le feedbacks et donc pour assurer le bon fonctionnement de la méthode.

5 Succès d'un Projet

Le succès d'un projet repose sur le coût qu'il a généré par rapport à la valeur du produit fournit, implique de la qualité.

6 Les Gens

La méthodologie est centrée sur les gens et ce qu'ils peuvent apporter à une équipe, sur leurs compétences, et attend d'eux une implication dans le projet.
Les gens ne sont pas juste des ouvriers à la chaîne remplaçables dans la minute comme dans ce que Taylor proposait, ils ne doivent cependant pas être indispensables au bon déroulement d'un projet (accident, etc...)

On attend des gens qu'ils prennent des décisions et proposent des estimations dans ce qui doit être fait, les managers doivent déléguer.

Chapitre 5

Agile

1 Création du terme

Agile vient du terme "Light" dans LightWeight.

Groupe de pratiques ayant pour base le **Agile Manifesto** rédigé en Février 2001 par 17 personnes conceptrices de méthodologies Lightweight, qui vont extraire les principes et critères de leurs méthodologies qui selon eux conduisent aux meilleures gestion de projets, afin de les réunir et créer le **Agile Manifesto**.
Ils créeront plus tard la **Agile Alliance**.

2 Manifeste - Valeurs

Les méthodes agiles prônent 4 valeurs fondamentales :
(Ctrl+C+V Wikipedia)

1. **L'équipe :** (« Les individus et leurs interactions, plus que les processus et les outils ») : dans l'optique agile, l'équipe est bien plus importante que les outils (structurants ou de contrôle) ou les procédures de fonctionnement. Il est préférable d'avoir une équipe soudée et qui communique, composée de développeurs (éventuellement à niveaux variables), plutôt qu'une équipe composée d'experts fonctionnant chacun de manière isolée. La communication est une notion fondamentale.
2. **L'application :** (« Des logiciels opérationnels, plus qu'une documentation exhaustive ») : il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est une aide précieuse mais non un but en soi. Une documentation précise est utile comme moyen de communication. La documentation représente une charge de travail importante, mais peut pourtant être néfaste si elle n'est pas à jour. Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).
3. **La collaboration :** (« La collaboration avec les clients, plus que la négociation contractuelle ») : le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un compte rendu continu sur l'adéquation du logiciel avec ses attentes.
4. **L'acceptation du changement :** (« L'adaptation au changement, plus que le suivi d'un plan ») : la planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premières livraisons du logiciel vont souvent provoquer des demandes d'évolution.

3 Manifeste - Principes

1. La plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.

2. Accueillir positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.

3. Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.

4. Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.

5. Réaliser les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leurs confiance pour atteindre les objectifs fixés.

6. La méthode la plus simple et la plus éfficace pour transmettre l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.

7. Un logiciel opérationnel est la principale mesure d'avancement.

8. Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commenditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
9. Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.

10. La simplicité, càd l'art de minimiser la quantité de travail inutile, est essentielle.

11. Les meilleures architectures, spécifications et conceptions émergent d'équipes autoorganisées.

12. A intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus éfficaces, puis règle et modifie son comportement en conséquence.

4 Agile - Répétition

Il faut répéter les opérations d'apparence fastidieuse telles que le testing ou le refactoring, celles-ci mènent à du feedback et à la volonté d'automatiser ces opérations.

5 Agile - Marketing

Contrairement à Prince2, la méthodologie Agile ne fonde pas tout un business autour d'elle-même.

Les labels ne sont pas importants, et le degré d'agilité n'est pas important non plus, tant que les méthodes mises en place contribuent au bon fonctionnement du workflow.

6 Agile - Excellence Technique

Seule l'excellence technique exigée permet ensuite de mettre en place l'Agilité.

Chapitre 6

SCRUM

1 Remarque

Scrum compile les bonnes pratiques de beaucoup d'autres méthodologies légères, et laisse la place à des idées venues d'autres horizons, même les vôtres éventuellement.

2 Origines

Métaphore liée à la mêlée du rugby utilisée pour la première fois dans une publication Japonaise de Hirotaka Takeuchi et Ikujiro Nonaka dans le monde industriel.

Cette publication va inspirer **Ken Schwaber** et **Jeff Sutherland** dans ce que deviendra la méthode SCRUM.

Ken Schwaber créera la Scrum Alliance pour proposer des certifications.

3 Rôles

On retrouve 4 rôles principaux dans un projet SCRUM :

3.1 Product Owner

Le Product Owner est la personne de référence pour la vision du produit, il est le **représentant du(es) clients**. Il est responsable des choix stratégiques liés aux fonctionnalités du produit, et a quattre tâches principales :

1. Définir les fonctionnalités : Gestionnaire des user stories en précisant leurs descriptions etc...
2. Répondre aux questions : Il fournit une vision partagée du produit, il énonce clairement l'énoncé du problème que le produit doit résoudre, s'assure que toutes les parties comprennent et partagent la même vision.
3. Planifier la vie du produit : Il définit l'ordre de réalisation des foncionnalités
4. Valider le résultat

3.2 Scrum Master

Ce n'est pas un rôle hierarchique, mais peu être apparenté au leader, il doit aider l'équipe à appliquer Scrum et à l'adapter au contexte.

Le Scrum Master gère et est reponsable du processus, il doit :

1. Veiller à la mise en application de Scrum en organisant les différentes réunions et en s'assurant qu'elles se déroulent dans le respect des règles établies conjointement par l'équipe
2. Encourager l'équipe à apprendre et à progresser
3. Faire en sorte d'éliminer ou de prévenir les obstacles qui pourraient freiner le processus (**impédiments**)
4. Inciter l'équipe à devenir autonome

3.3 Team

Est inter-fonctionnelle, pas d'ultra spécialisations, chaque membre peut aider un autre sur n'importe qu'elle partie du projet à tout moment.

L'équipe doit trouver ses propres solutions aux problèmes via des initiatives et des prises de décisions.

3.4 Stakeholders

Toute personne ayant un intérêt dans le résultat (financier, scientifique, etc...)

4 Processus

Le Produt Owner commence par établir un Product Backlog autour duquel l'équipe va organiser le planning d'un Sprint d'une certaine durée fixe.

Chaque jour, l'équipe se réuni quelques minutes pour faire un Daily Scrum.

A chaque fin de Sprint, l'équipe se réuni pour faire un démonstration Sprint Review de ce qui a été produit de fonctionnel, et faire une rétrospective de Sprint pour discuter de ce qui pourrait être amélioré.

4.1 Sprint Planning

Team, Product Owner et Scrum Master se réunissent pour débuter le Sprint.
Chaque tâche du Backlog va être estimée en temps grossièrement via un planning poker, et un Selected Product Backlog va être produit en comportant toutes les Stories que la Team s'engage à remplir.

Ce Selected Product Backlog avec les tâches estimées en points formera le Sprint Backlog.

4.2 Daily Scrum

En début de journée, la team et le Scrum Master se réunit pendant 15 minutes pour expliquer :

1. Ce qu'il a fait hier
2. Ce qu'il va faire aujourd'hui
3. Ce sur quoi il a des problèmes

Le Product Owner peut éventuellement assister à ce Daily Scrum mais n'a pas le droit de participer, juste de s'informer.

4.3 Sprint Review Meeting

En fin de Sprint, la trinité se réunit avec les clients et les stakeholders et va présenter les Stories terminées devant l'assistance.

De cette manière, l'équipe va pouvoir avoir le feedback de ceux-ci et mettre à jour le Backlog.

4.4 Sprint Retrospective Meeting

L'équipe et éventuellement le Product Owner se réunissent pour discuter de :

1. Ce qui a bien été
2. Ce qu'il faudrait améliorer ou changer

4.5 Backlog Refinement

Cette étape n'est pas forcément effectuée au début ni tout court.

Elle consiste, en milieu de sprint, a repasser le Backlog et a éventuellement revoir certaines fonctionnalités, afin d'estimer comment va se passer la suite.

5 Definition of Done

L'équipe doit se mettre d'accord sur la définition de ce qui est fini, par exemple un élément finit pourrait avoir les caractéristiques suivantes :

— Inclus dans le source control
— Couvert par des tests unitaires
— Couvert par des tests d'intégration
— Documenté et màj

Ceci est très utile pour l'estimation des tâches et afin de terminer et finir tout correctement.

6 Burn Down Chart

Estime le temps sur un Sprint, mais trop rigide car inadapté aux changements si réestimation de temps ou ajout de temps de travail.

Désavantage au niveau du client, qui verra presque forcément un non-respect de ce planning qui n'est censé être là qu'à titre informatif pour l'équipe.

Une analyse a montré que le Burn Down Chart avait presque toujours la même forme pour un Sprint, lent démarrage puis gros boost en fin de Sprint.

7 Burn Up Chart

8 Chicken n Pigs

9 Transparence

Scrum demande du travail en transparence par rapport à toutes les personnes impliquées, ceci est assuré via :

— **Daily Scrum** : Tout le monde est au courant de ce que tout le monde fait, pas de secrets
— **Sprint Review** : Les clients sont régulièrement mis au courant de l'état dans lequel se trouve le projet, impossible d'essayer de lui cacher que ça n'avance pas très bien ou qu'il y a eu tel problème
— **Burn Down Charts, Kanbans, etc..**

Tout cette transparence mène sur des prises de décisions meilleures.

10 Remarques

Le bon fonctionnement de Scrum a pour condition **l'engagement** de chaque personne de la team.

Bien que les méthodes légères veulent se défaire de plannings, un planning de Sprint en tant que feedback et non en tant que ligne conductrice n'est pas forcément mauvais.

Les idées sont plus importantes que le vocabulaire, il faut comprendre les idées que l'on veut appliquer.

11 Tableau des Tâches

Le Kanban vient de Mike Cohn et permet via un système de **Post-It** de placer les éléments du BackLog sur un tableau et de les répartir dans trois colonnes :

— Todo
— Doing
— Done

Permet un gain au niveau transparence et transmission des idées au sein de l'équipe.

12 Planning Poker

Jeu de cartes reprenant la suite de Fibonacci, tout le monde choisi une carte pour estimer une tâche et la montre en même temps, ensuite il y a discussion.

Fibonacci permet d'avoir des valeurs assez éloignées les unes des autres, évite les discussions pour un écart faible, provoque vite un consensus.
Permet également d'avoir l'avis des personnes qui auraient une valeur assez extrême sur sa carte par rapport aux autres et d'apporter un point de vue peut-être essentiel que les autres auraient oublié.

Enfin, permet d'identifier les tâches trop grosses qu'il faudrait sans doute raffiner.

Chapitre 7

Extreme Programming

1 Origines

Méthodologie Agile créée par Kent **Beck**, Ward **Cunningham** et Ron **Jeffries** lors d'un projet chez Chrysler.

En 1999, le premier manifeste est édité avec 12 pratiques.

2 Valeurs

XP possède 5 valeurs qui sont toutes reliées entre elles :

— Courage
— Simplicité
— Communication
— Feedback
— Respect

2.1 Simplicité

Leitmotiv : **You Ain't Gonna Need It**

Ne pas faire quelque chose si on est pas certain d'en avoir besoin, cela risque de demander plus de travail si jamais on venait à en avoir besoin, mais dans le cas contraire on a pas gaspillé d'argent et cela a permis de se concentrer sur des choses essentielles.

2.2 Communication

La communication doit se faire de manière **orale**, il doit y avoir une collaboration avec les utilisateurs et les connaissances doivent être partagées.

On évite au maximum la communication par écrit.

2.3 Feedback

Le FeedBack est assuré par le Unit Testing, les démos, on encore les estimations, qui apportent tous une amélioration aux projets.

2.4 Respect

Les deux valeurs précédentes nécéssitent le respect de tout les gens impliqués dans le processus.

2.5 Courage

Il faut avoir le courage de changer, de jeter, d'être transparent et ne pas essayer de cacher les choses ni de se protéger lorsqu'il y a un problème, l'équipe est un tout.

3 Unit Testing

Le test unitaire est, comme son nom l'indique, un test qui va vérifier le bon fonctionnement d'une petite partie d'un programme.

Ces tests sont écrits au fur et à mesure du développement, en essayant de couvrir le plus de cas possibles selont les fonctionnalités.
Chaque itération sur le développement doit toujours pouvoir passer les tests qui ont été écrits plus tôt, si une opération doit retourner tel résultat au début du projet, elle doit toujours le faire à la fin.

Ceci va amener beaucoup de points positifs dans le développement. Il n'y a plus de régression dans le code car on ne peut faire qu'avancer malgré les modifications, on explore les **cas limite** des fonctionnalités, on a plus peur de **changer** car on saura contrôler les effets du changement sur la base, et enfin le code est **modulaire**.

Le unit testing prend du temps, et le temps c'est de l'argent, mais celui-ci devient malgré tout très vite rentable.

La règle : **Le code doit passer TOUT les tests unitaires avant sa release**.

4 Tests d'Acceptance - Fonctionnels

Rédigés avec le client, trace une ligne conductrice qui définit les critères que doit remplir l'application selon les User Stories.
Il consiste plus ou moins à répertorier les actions que le client va effectuer pour voir si le programme est fonctionnel.

Le client peut dire qu'il veut faire une addition à tel endroit et que cela doit lui ramener tel résultat, on lorsqu'il clique à tel endroit dans une app web cela doit avoir telle action.

Ces tests sont écrits et ensuite automatisés par un programme de sorte que le développeur puisse savoir si le programme remplit bien les tests d'acceptance.

Ces tests prennent du temps, la maintenance est assez lourde, mais le client peut savoir très facilement si le programme fait bien ce qu'il a demandé.

5 Test-Driven Development

Le principe est d'écrire les tests avant même d'écrire le code qu'il couvre.

Boucle d'états : **Red** - **Green** - **Refractor**, jusqu'à ce que le code soit propre et en état Green.

6 Exploratory Testing

On ouvre l'application pour tester tout et n'importe quoi sans trop réfléchir.

Toujours utile et complémentaire, mais doit être laissé à un non-développeur qui ne connaît pas les rouages de l'application.

7 Refactor Mercilessly

Le **refactoring** mène à un meilleur code pour de mêmes fonctionnalités.

Il réduit la complexité de celui-ci et facilite beaucoup les évolutions future.

8 Small Releases - Iterations

XP fonctionne sur base de petites releases et non pas une unique finale, les itérations durent de 1 à trois semaines.

Le **Release Planning Meeting** se fait avec l'équipe et le client, on y discute de la durée de l'itération et des fonctionnalités qui vont être implémentées.

L' **Iteration Planning Meeting** se fait entre l'équipe, les tâches sont évaluées en termes de complexité et de temps afin d'organiser l'itération.

9 Velocity

L'important est de mesurer la vélocité de l'équipe et non pas la calculer, ceci dans le but d'avoir un point de repaire et se stabiliser sur une somme de travail que l'équipe **PEUT** effectuer en une itération.

On discute de points et non pas d'heures, l'estimation d'heure dépend des personnes, sont souvent sous-estimées et on a peur des gros chiffres. L'estimation en points est relative et arbitraire.

10 Sustainable Pace

Le but de la quantification en points est de proposer à l'équipe un rythme de travail tenable et confortable, pour garder une bonne motivation au sein de celle-ci.

11 Stand-Up Meetings

Séance très courte en début de journée, tout le monde dit ce qu'il a fait hier, ce qu'il va faire aujourd'hui et ce qui lui pose problème.

12 User Stories

Les fonctionnalités sont décrites en quelques lignes de texte, l'objet est concret (post-it), peu de détails car il doit servir à amener la discussion, vocabulaire user et permet une rapide estimation.

Elles doivent être **INVEST** :

— **Independant** : Ne dépend pas d'autres stories
— **Negotiable** : Peut toujours être modifiée
— **Valuable** : Fournit de la valeur à l'utilisateur
— **Estimable** : Possible de faire une estimation
— **Small** : Limite l'incertitude de l'estimation
— **Testable** : Possible de tester que c'est fait ou non

13 Epics

Regroupe un ensemble de user stories.

14 Integrate Often

L'intégration des différentes portions du programme doit être continue, sinon on risque de se retrouver avec un **Integration Hell** : pas moyen de savoir à quel moment du projet un problème a été créé.

Cette intégration continue amène du Feedback tôt, de la confiance et surtout un produit fonctionnel tout au long du projet.

Des programmes Cloud ou locaux peuvent servir à effectuer cette intégration en automatisant le testing, et donc en vérifiant qu'une feature ne pose pas problème dans une autre.

15 Radiateurs d'informations

Présente les états de build, change requests et autres de manière claire et visible à l'équipe.

16 Big Visible Charts

Typiquement le Kanban Agile, il sert surtout à la communication et à la transparence, quiconque peut voir où en est l'équipe et savoir si des imprévus sont survenus.

17 Coding Standards

L'équipe doit s'accorder sur un standard de codage afin de produire un code uniforme et facilement maintenable et opérable par n'importe quelle personne de cette équipe, le standard ne doit pas être imposé.

18 Collective Ownership

La production appartient à tout le monde et pas seulement à l'architecte.

19 CRC Cards

Sorte de cartes représentant une classe, utilisées pour discuter de scénarios et mettre en évidence des attributs ou méthodes qui n'auraient pas été pensés.

Class, **R**esponsibilities, and **C**ollaboration cards.

20 Pair Programming

Deux programmeurs se mettent ensemble, tels un pilote et un copilote, le pilote écrit sur le clavier et le copilote monitore ce que l'autre écrit.

Ces deux personnes sont considérées égales, peuvent changer de rôle.

Ceci va permettre d'éviter certains bugs liés à de l'inatention.

21 Simplicity Is The Key

22 Never Add Functionnality Early

23 Create a Spike Solution

Consiste en la création d'un Proof of Concept afin d'évaluer si ce projet est faisable avec telle ou telle nouvelle techno ou contraintes spéciales.

Cette SPike Solution doit être jetée quoiqu'il arrive, même si l'équipe décide que le Spike donne le feu vert pour le projet, les éléments du Spike ne peuvent pas être insérés dans le projet.

24 Choose a System Metaphor

Métaphore type ruche pour les échanges de données avec le système.

25 Technical Debt Metaphor

De la précipitation amène en premier lieu un grain de temps.

Ce gain de temps doit être ensuite payé par du refactor et de la correction de bug.

26 Dedicated Open Workspace

L'OpenSpace est une pièce ouverte sans murs intérieurs où on peut afficher des graphiques, un tableau blanc, il y a une table de discussion et de la place pour des stand-up meetings.

27 Move People Around

Il faut changer les membres de l'équipe de zone, afin qu'ils acquièrent de multiples compétences, qu'ils fassent le lien entre les systèmes, qu'on diminue le risque de problème fatal en cas de perte d'un membre, ou qu'on évite des goulots d'étranglement sur certains points du programme.

Idée précurseur de DevOps ?

28 The Customer Is Always Available

Difficile à réaliser en vrai, le client devrait être toujours disponible pour l'équipe.

Il est difficile de faire comprendre au client que la perte de temps qu'il subit en intervenant en direct dans le process est largement rattrapée par les problèmes qui ne sont pas rencontrés en évitant la communication par écrit.

29 Fix XP when it Breaks

XP est une base pour l'équipe qui doit évoluer en fonction de ses besoins, nécéssité de faire des rétrospectives de Sprint.

Chapitre 8

Behavior Driven Development

Consiste en du TDD amélioré.

Les tests portent un nom qui signifie le contenu du test et est compréhensible par le commun de mortels, on sait également déterminer le type d'erreur lorsqu'un test échoue.

Syntaxe **Give**, **When**, **Then**.