Université Paul Sabatier – Toulouse III
IUT A - Toulouse Rangueil

Processus de développement Agile

Semestre 4

Toulouse, le 27 février 2012

Avant-propos – Les méthodes agiles : une réponse à un malaise ?

Un étude à montré que les projets informatiques ont tendance à ne pas aboutir ou à couter beaucoup plus cher que prévu :
- 25% des projets seulement respectent les délais et les budgets
- 45% dépassent le budger ou sont en retard
- 28% sont abandonnés ou voient leur périmètre largement restreint
- Combien correspondent à des besoins utilisateurs rééls ?!

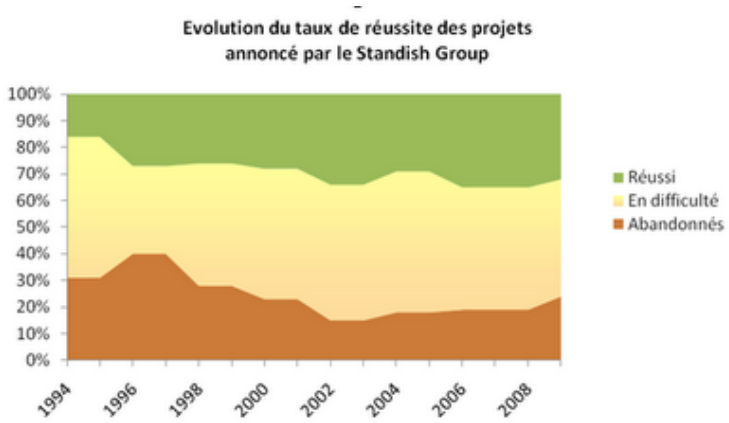

FIGURE 1 – Taux de réussite des méthodes classiques

Les causes de ces echecs

Afin de comrendre comment les méthodes Agiles essayent de répondre à ces problèmes, il faut d'abord se demander, pourquoi ces problèmes existent.

La conformité aux besoins Les projets ne respectent souvent pas les besoins du client, ou ne comprennent pas ces besoins. Ainsi c'est après des centaines d'heures de développement que le client affirme que ce n'est pas ce qu'il voulait.

Les technologies La technologie choisie pour le projet est extrememnt importante, celle ci doit être capable de permettre de respecter les besoins du client, l'équipe doit la maîtriser, elle doit permettre d'effectuer un développement le plus rapide possible pour ce projet.

Les méthodes Les méthodes de développement vont elle aussi également grandement influencé l'avancée du projet, certaines méthodes peuvent permettre de résoudre les problèmes aux plus tôt, mais elle doit être adaptée à la taille du projet.

Les facteurs clé de l'échec

Les méthodes de développement classiques ont des ponits commun quant à leur raison d'échec.
- Le manque de communication à tout niveau
- Une mauvaise compréhension des besoins
- L'insuffisance de l'architecture
- L'absence de maturé des outils utilisés
- La mauvaise formation des personnes
- Le cadre contractuel inadapté
- L'insuffisance des tests

Les facteurs aidant à la réussite

Le Standish Group à identifié une dizaine de conditions de succès favorisant la livraison de projets informatiques fructueux. Par ordre d'importances, ces conditions sont :

1. L'engagement de la direction
2. L'implication des utilisateurs
3. L'expérience du chef de projets
4. La formulation des objectifs d'affaires
5. Une envergure limitée aux besoins essentiels
6. Une infrastructure technologique et stables
7. Des métholodogies formelles et utilisées
8. Des estimations fiables et rigoureuses
9. Autre : découpage des livraisons, compétence du personnel, etc...

Quelques explications de l'échec

Le client ne maîtrise pas le projet Le premier problème vient du fait que le client ne maîtrise rien. Il ne connait pas le langage de modélisation utilisé, et se retrouve en conséquence exclue du processus dès la fin des interviews. Également, l'équipe ne va pas lui montrer régulièrement une avancée du projet, or, c'est lui qui sait ce qu'il veut, il pourrait alors mieux préciser ces besoins et les problèmes que peuvent comporter le projet au fur et à mesure.

Cycles de développement trop long Avoir des cycles de développement court permet de mieux maîtriser l'avancée du projet, tester régulièrement [1] permet d'éviter de gros prolbèmes de conception.

1. Éventuellement avec du développement dirigé par les tests

Table des matières

CHAPITRE 1

Généralités sur les méthodes Agiles

1.1 Historique

En 1968 à eu lieu la naissance du Génie Logiciel, nous sommes passé de la programmation non structurelle à la programmation structurelle avec Cobol et Fortran Cela à permis d'avancer, avec de nouveaux langages, des OS principalement mais aussi avec :

Les premières méthodes méthodes remontent à 1994 avec Scrum et 1996 avec l'Extreme Programming, de nos jours, les deux disciplines ont tendances à se rapprocher.
- Le cycle en V (Aproche séquentielle)
- L'hyper-management (Taylorisme)
- L'hyper-formalisme (exigences, conception)
- L'obsession de la mesure et métrique

Le terme "Agile" apparaît en 2001 (Snowbird, rédaction du "manifest"), il est apparut pour répondre aux malaises énoncés en Avant-propos page i.

Dans les méthodes Agile on essaye de mettre l'humain et le relationnel au centre du développement. Par exemple les problèmes doivent être résolus en face à face. Le **client** est au centre du développement, on part de l'idée que le client va changer d'idée, et donc qu'on le contacter régulièrement pour connaître son avis, et ses éventuels changements.

Elles ont également un ensemble de règles, que l'équipe se doit de respecter.

Ex Extreme Programming

- Le code au centre de tout
- Programmation en binome (Importance de la qualité)
- Tests unitaires
- Meeting réguliers (tous les jours)
- Tests de validation
- Iterations

1.2 Caractéristiques des méthodes Agiles

Il existe une multitude de méthode Agiles différentes, cependant, elles ont toutes les même caractéristiques et le même but d'avoir un projet aboutit qui aura couté le moins cher possible.

1.2.1 L'humain est au centre du développement

En effet, dans les méthodes Agiles, l'Humain est le plus important, autrement dit, **le client** doit être vu très régulièrement, pour qu'il puisse donner son avis sur le projet, et éventuelleemnt donner l'avancée de sa réflexion sur ses besoins.

Également, l'**équipe** doit être soudée, et investit dans son projet, elle doit participer d'un bout à l'autre du projet, il est ainsi indispensable d'avoir des équipes petites afin d'éviter aux développeurs de se sentir noyés dans la masse.

1.2.2 La qualité au centre du développement

Avoir un travail de qualité induit donc que le programme final devra correspondre au client comme expliqué en section 1.2.1, mais également ne doit pas être bugué, ainsi les **tests** sont extrêmements important.

Le programme devra lui aussi avoir été codé avec le plus grand soins, afin d'avoir un **code propre** qui permettra une maintenance facile, pour cela certaines méthode Agiles préconisent la programmation en binôme.

1.2.3 Méthodes Agiles Versus Méthodes classiqus

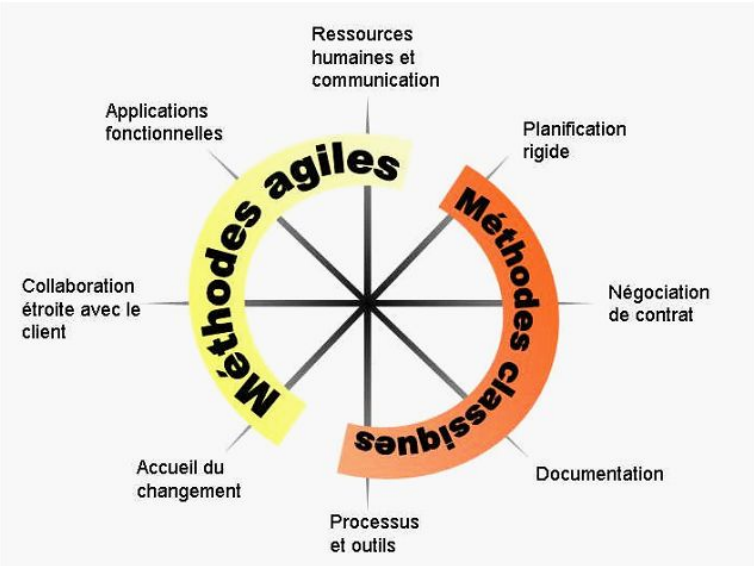

FIGURE 1.1 – Caractéristiques des méthodes Agiles

1.3 Panorama de différentes méthode sagiles

Nous allons étudier très rapidement les différentes méthodes Agiles qui peuvent exister, ce sont les méthodes Agiles les plus utilisées :
- UP
- eXtreme Programming
- Dynamic Software Development Method
- Adaptive Software Development
- Crystal Clear
- SCRUM

1.3.1 UP

Les fondamentaux d'UP :
- Pilotage par les cas d'utilisation
- Centré sur l'architecture
- Itérative et incrémentale
- Gère les besoins et les exigences
- Fondée sur la production de de composants
- Pratique la modélisation visuelle
- Surveille la qualité et les risques

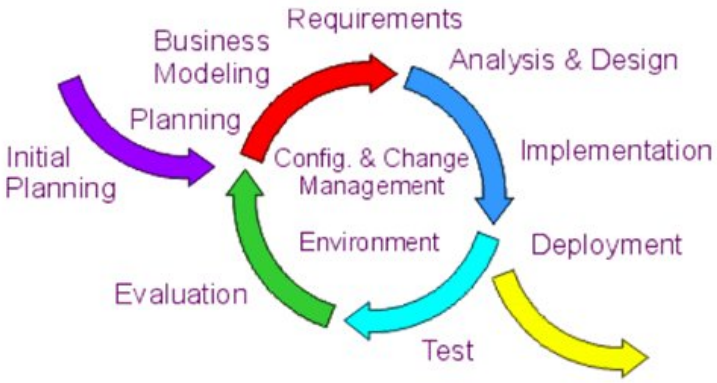

FIGURE 1.2 – Cycle de vie de UP

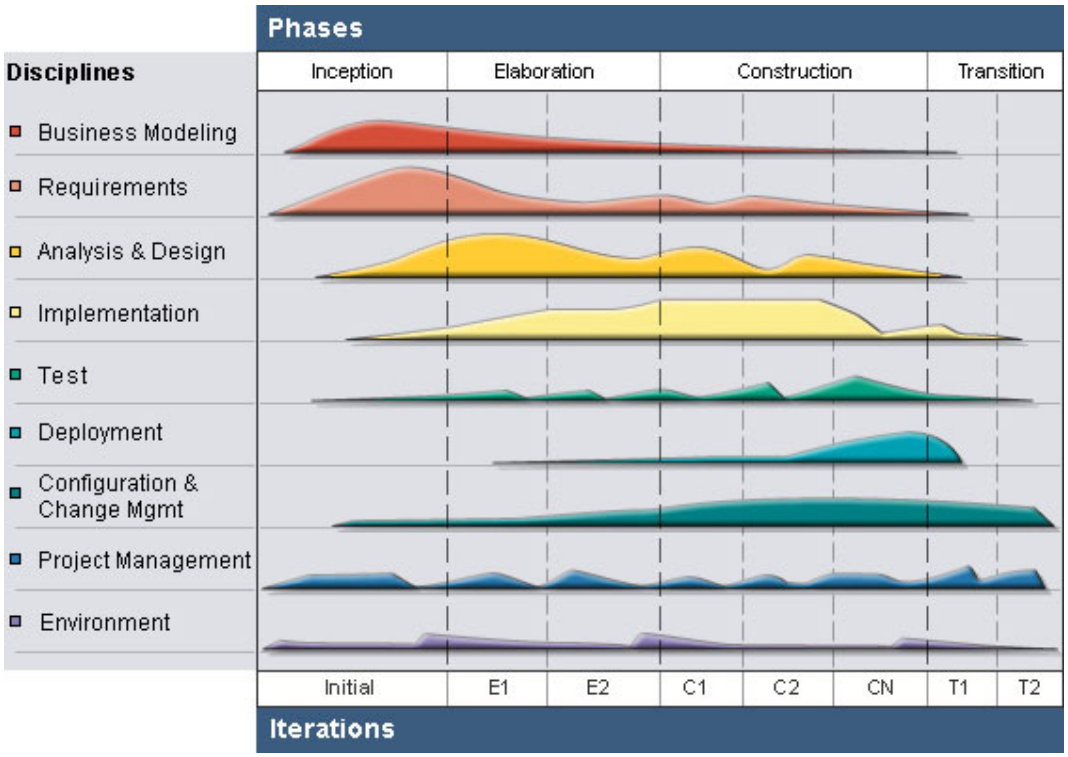

FIGURE 1.3 – Phases de développement de UP

1.3.2 eXtreme Programming

Les créateurs sont Ward CUNNINGHAM et Kent BECK.

1.3.2.1 Valeurs clefs

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

Feedback Indispensable pour que le projet puisse accueillir le changement

Courage Concerne aussi bien les développeurs que le client

1.3.2.2 Quelques pratiques

- Planning game
- Petites releases
- Métaphores pour décrire l'architecture
- COnception simple

– Tests
– Refactoring
– Programmation en binôme
– Propriété collective
– Intégration continue
– Pas de surcharge de travail
– Client sur site
– Standars de code (conventions)

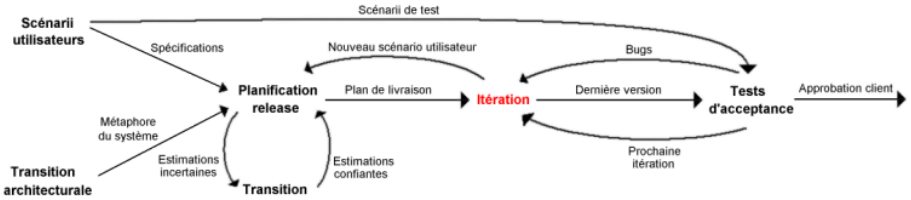

FIGURE 1.4 – Cycle de vie de eXtreme Programming

1.3.3 Dynamic Software Development Method

DSDM [1] possèdent à peu près les même principes que les autres méthodes. Elle trouve ses origines en Angleterre au début des années 1990. L'idée fondamentale derrière cette méthode est de fixer le temps et les ressources et de faire varier le nombre de fonctionnalités en conséquence.
– Implication actives des utilisateurs
– Equipes autorisées à prendre des décisions
– Produit rendu tangible aussi souvent que possible
– L'adéquiation au besoin métier est le critère essentiel pour l'acceptation des fournitures
– UN dévelopement itératif et incrémental permet de converger vers une solution appropriée
– Toute modification pendant la réalisation est réversible
– Besoins définis à un niveau de synthèse
– Tests intégrés pendant tout le cycle de vie
– Esprit de coopération entre tous

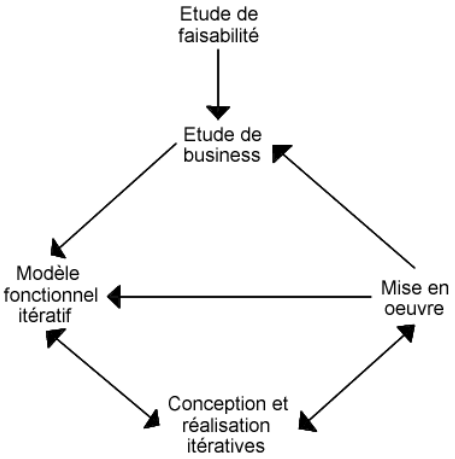

FIGURE 1.5 – Cycle de vie de DSDM

1. Dynamic Software Development Method

1.3.4 Adaptive Software Development

La méthode ASD [2] à été crée par Jim HIGHSMITH, elle possède 6 caractéristiques principales :
- Mission focused
- Component-based
- Iterate
- Timeboing
- Projet risk-driver
- Responding to change

1.3.4.1 Spéculer

- Initier le projet
- Déterminer la date butoir du projet
- Définir le nombre d'itérations et les dates associées
- Donner une missions à chaque itération
- Affecter les composants de base aux itérations
- Affecter les technologies aux itérations
- Développer une liste de tâche à réaliser

1.3.4.2 Collaborer

- Délivrance des composants
- Communication forte et assez informelle
- Possibilité d'apliquer des pratiques Agiles (de type XP)

1.3.4.3 Apprendre

- Contrôle qualité utilisateur / client
- COntrôle qualité technique
- Suivi de la performance
- Bilan sur l'état d'avancement
- COmmunication forte et assez informelle
- Possibilité d'appliquer des pratiques Agiles (de type XP)

1.3.5 Crystal Clear

Méthode créée par Alistair COCKBURN.

Les points clefs
- Importance de la communication et du feed-back client
- Releases fréquentes
- Deux grandes phases
  - Conception et planning

2. Adaptive Software Development

  - Itérations
- Adapté à des équipes de 6 personnes maximum (pas de leadership clairement exprimé)

1.3.6 SCRUM

La méthode scrum est expliqué en détaille au chapitre 2 page 8.

1.4 Synthèse

Les méthodes agiles semble adaptés à un besoin :
- Pas de sous-traitance
- Équipe motivée
- Équipe légère
- Équipe peut nombreuse
- Environnement non critique
- Projet non complexe

1.4.1 Comparaisons entre les méthodes

Méthodes	**Dites classiques ou "lourdes"**	**Dites nouvelles ou "Agiles"**
Paradigme	Prédictivité	Adaptabilité
Fondement	Analytique cartésien	Pragmatique empiriste
Cycle projet	EN cascade	Incrémental et itératif
Risque	Descriptive	Expérimentation
Raisonnement	Discursifs	Systémique
Pensée	Réductionniste	Vision holistique
Philosophie d'analyse	Considère la nature des intéractions	Considère les effets des intéractions
Axe de recherche	L'analyse de la structure	L'aboutissement des actions
Conduit à des systèmes	À forte entropie	À forte rétroactivité
Philosophie d'action	Conduit à une action totalement détaillée et programmée	Conduit à une action par objectifs et flexible
Aboutit concrètement	À la reconduction de la structure existante	À l'amélioration des éléments de performance
Validation	Comparaison théorique à base de jeu d'essais en fin de parcours	Confrontation permanente du modèle avec la réalité (prototype)

CHAPITRE 2

Scrum

2.1 Rôles

La méthode Scrum dispose de plusieurs rôles, chacun des rôles est important, en effet, comme toutes les méthodes Agiles l'humain est au centre du projet.

Le Product Owner C'est le client, il est extrememnt important, l'équipe de projet doit le voir régulièrement pour qu'il valide chacun des sprints [1] afin de voir si ses besoins sont respectés.

Le Scrum master Il aide l'équipe il va essayer d'aider au mieux le déroulement du projet et veiller au respect de Scrum, en cas de problème, c'est lui qu'il faut contacter.

L'équipe Ce sont toutes les personnes développant le projet. Elle ne doit pas trop nombreuse pour que les développeurs se sente impliqués dans le projet, et qu'ils puissent avoir une vision globale du projet.

2.2 Timeboxes

Timeboxes Une boîte de temps ou "timebox" est une période fixe pendant laquelle on cherche à exécuter le plus efficacement possible une ou plusieurs tâches.

2.3 Story

Une story est exprimée sous la forme suivante :
En tant que ...
Je désire ...
Quand ...
Afin de ...
Cette story va avoir un certain nombre d'état.

1. Les sprints sont définis section 2.4 page 9

2.3.1 Éléments composants la story

- Un nom
- Une description
- Un Type

  Ex User, Default, Technique...

- Un État

  R Les états peuvent être les suivants, une story doit passer par tous les états.
  1. Créé (Tout le monde)
  2. Acceptée (Product Owner)
  3. Estimée (Équipe)
  4. Planifiée (Équipe)
  5. En cours
  6. Finie (Prend en compte les tests)

- Un Poids

  Ex 1, 3, 5, 8, 13 ...

  Le poids est basé sur la suite de Fibonacci

- Un BVP [2], cela sert à connaître la priorités des stories

  R
  - (900–1000) Must (indispensable)
  - (700–800) Should
  - (400–600) Could
  - (100–300) Would

  La priorité est donnée par le client.

2.4 Sprints

Un sprint correspond à un incrément du projet. En effet, avec la méthode Scrum nous travaillons par incrément, le but étant d'avoir un programme fonctionnelle à la fin de chaque sprint. Ainsi, nous devons pouvoir montrer une démonstration au Product Owner, chaque sprint devra être une surcouche du précédent et ainsi améliorer le logiciel, tout en restant totalement fonctionnel.

Un sprint contient plusieurs stories qui devront être finies à la fin du sprint.

R Une tâche est finie que lorsque celle-ci à été correctement testée, que celle-ci est fonctionnelle, et peut donc être montrée au client.

2. Business Value Point

2.5 Revue

Une revue permet un feedback concret sur le produit (c'est mieux que sur de la documentation).

Le but de la revue est de montrer ce qui a été réalisé pendant le sprint afin d'en tirer les conséquences pour la suite du projet.

2.5.1 Étapes

Voici les différentes étapes de la revue de sprint.
- Sprint basé sur une démo
- Calculer la vélocité de l'équipe.
- Release
- Ajouter les autres sprints et la release

À la fin des plans de sprint et des plans de release, on calcule :
- La vélocité
- La capacité

2.6 Artefacts

2.6.1 Backlog produits

Les équipes agiles ne produisent pas une documentation faite au début du projet, qui décrit en détail toutes les spécifications fonctionnelles. Elles collectent les fonctions essentielles (les features) et les raffinent progressivement. Il n'y a pas un gros document de spécification, l'outil de collecte s'appelle le backlog de produit.

2.6.1.1 Objectif

Le backlog de produit est la liste des fonctionnalités attendues d'un produit. Plus exactement, au-delà de cet aspect fonctionnel, il contient tous les éléments qui vont nécessiter du travail pour l'équipe. Les éléments y sont classés par priorité ce qui permet de définir l'ordre de réalisation.

2.6.1.2 Rôle

Le backlog de produit est sous la responsabilité du product owner. Chacun peut contribuer à collecter des éléments, mais c'est le product owner qui les accepte finalement et c'est lui qui définit les priorités.

2.6.2 Plan de sprint

Les plans de sprints correspondent à ce qui est prévu pour le sprint. C'est-à-dire les différentes stories que nous souhaitons développer pour l'increment. Ces stories doivent être regouprés en fonction de leurs poids et de leurs BVP.

2.6.3 Plan de releases

Le plan de release correspond au plan de la globalité du projet. C'est-à-dire les plans de chaque sprints.

2.6.4 Burndown

L'équipe affiche en grand format, sur un des murs de son local, un graphe représentant la quantité de travail restant à effectuer (sur l'axe vertical) rapportée au temps (sur l'axe horizontal). C'est un "radiateur d'information".

Ce graphe peut concerner l'itération en cours (“iteration burndown”) ou plus couramment l'ensemble du projet (“product burndown”).

ANNEXE
A

Signification des acronymes

AGL Atelier de génie logiciel

MOA Maître d'ouvrage (client)

MOE Maître d'œuvre

RE Requirements Engineering

SWAT Skilled With Advanced Tools

UC Use Case

US User Story