Tendances des méthodes de gestion des projets informatiques

Lamia Ben Hiba

ENSIAS, Université Mohammed V-Souissi, Madinat Al Irfane, Rabat, Maroc
benhiba.lamia@gmail.com

Mohammed Abdou Janati Idrissi

ENSIAS, Université Mohammed V-Souissi, Madinat Al Irfane, Rabat, Maroc
janati@ensias.ma

Résumé

Afin de faire face à la complexité ascendante du développement logiciel, des méthodes traditionnelles de gestion des projets informatiques ont été introduites. Axées-plan et généralement lourdes, ces méthodes ont perdu de plus en plus de partisans en faveur des nouvelles méthodes dites « agiles ». Cependant, ces dernières présentent des limitations pour les projets critiques ou ceux avec une équipe de grande taille. Ainsi, ont vu le jour de nouvelles méthodes dites « hybrides », palliant les faiblesses et bénéficiant des forces des deux catégories de méthodes. Cet article présentera différentes approches traditionnelles, agiles et hybrides de gestion de projet informatique, dressera un comparatif entre les méthodes traditionnelles et agiles et enfin discutera du problème du choix de la méthode de gestion à adopter pour un projet donné.

Abstract

Traditional methods were introduced in order to face the ascending complexity of software development. Plan-oriented and generally heavy-weighted, these methods have been loosing advocates in favour of new methods called Agile. These methods present certain limitations when it comes to critical projects and big teams. To benefit from both methods' strengths and to palliate their weaknesses, Hybrid methods appeared. This paper aims to present different traditional, agile and hybrid project management methods, provide a comparison between traditional and agile methods and discuss the problem of the choice of the method to adopt for a given project.

Mots-clés

gestion de projet, Pmbok, CMMI, méthodes Agile, ASD, DSDM, Scrum, XP, modèle de Boehm et Turner, modèle de Sidky

Keywords

Project managment, Pmbok, CMMI, Agile method, ASD, DSDM, Scrum, XP, Boehm et Turner's model,Sidky's model

1. Introduction

L'industrie logicielle a intégré la société moderne depuis plus de 50 ans. Au début, le développement logiciel n'était que le résultat d'activités désordonnées (*cod and fix*) sans planification, ni conception détaillée en amont. Cependant, la complexité croissante des logiciels a rendu difficile l'ajout de nouvelles fonctionnalités ou la réparation des *bugs* systèmes. Ainsi, fallait-il impérativement introduire des méthodes de développement logiciel. Les méthodes sont venues imposer un processus discipliné avec pour objectif de rendre le développement logiciel plus prédictible et efficace.

Les méthodes traditionnelles sont des approches axées-plan qui se basent sur un ensemble d'activités séquentielles. Un des principaux avantages de ce modèle est la réduction des risques grâce à la mise en place de vérification et de validation à la fin de chaque étape. Cependant, l'utilisation reste lourde et contraignante avec notamment une documentation conséquente et difficile à rédiger, ainsi qu'un client exclu des vérifications des différentes phases. Perçue comme rigide et limitant la créativité, l'approche traditionnelle a été remise en cause par de nouvelles méthodes dites agiles, l'objectif étant d'introduire de la flexibilité au sein des organisations. Légères, adaptatives au changement, ces méthodes ont gagné en notoriété. Pourtant, on leur reproche de ne pas être adéquates aux projets complexes ou critiques.

Partant du constat que les méthodes agiles et traditionnelles ne sont pas incompatibles, de nouvelles approches hybrides ont émergé afin d'intégrer les processus agiles dans le cycle de développement traditionnel. Ces approches sont assez récentes (début des années 2000) et elles ont donc encore besoin de perfectionnement. Cependant, la plupart d'entre elles ont contribué à déterminer la disposition des organisations à aller vers l'agilité et ainsi ont permis l'énumération de quelques facteurs influençant le choix des méthodes.

Le présent article expose un état de l'art des méthodes de gestion de projets informatiques. Nous présentons en premier lieu les principes des méthodes traditionnelles en citant quelques méthodes. La deuxième section traite des méthodes agiles, leurs principes et leurs limites en dressant une comparaison entre les principales méthodes existantes. Une comparaison entre les méthodes agiles et les traditionnelles fait l'objet de la troisième section. La quatrième section décrit les approches hybrides. Nous abordons dans la dernière section la question du choix de la méthode adéquate pour la gestion des projets informatiques.

2. Méthodes axées-plan (traditionnelles)

2.1. Principes des méthodes axées-plan

Les méthodes axées-plan sont considérées comme étant le moyen traditionnel pour le développement logiciel. Fondées sur des concepts tirés des principaux domaines de l'ingénierie, ces méthodes perçoivent le développement selon un paradigme de besoins/conception/développement couplé de processus standards et bien définis que l'organisation améliore constamment. Plusieurs cycles de vie ont été créés se fondant sur cette approche. Le plus célèbre est le cycle de vie en cascade introduit par Royce (1970), où les phases de développement sont implémentées (au moins deux fois) l'une après l'autre. Même si ce cycle de vie est considéré comme étant le modèle le plus axé-plan qui existe, Royce suggère que pour réaliser un projet nécessitant une durée d'exécution de 30 mois, on doit d'abord réaliser un projet-pilote de 10 mois, ce qui induit un aspect itératif au modèle. Ainsi, le système et le développement logiciel ne sont pas des processus linéaires. Boehm (1988) conclut que suivre le cycle de vie en cascade conduit à une approche axée-documents, ce qui force les membres de l'équipe développement à écrire des documents inutiles et difficiles à comprendre. Ainsi, Boehm présenta le cycle de vie en spirale, nouveau modèle itératif, pour pouvoir naviguer vers chacune des phases du cycle de développement et faciliter la communication et la confiance entre les membres de l'équipe. Dans les années 90, de plus en plus de méthodes itératives et/ou incrémentales ont vu le jour, mais elles maintenaient une lourde documentation et une traçabilité minutieuse à travers l'analyse des besoins, la conception et le développement.

Dans le chapitre qui suit, nous présentons deux exemples de méthodes axées-plan qui manifestent les aspects des méthodes traditionnelles. Nous commençons par le célèbre guide PMBOK (Project Management Body of Knowledge). Quoiqu'il ne représente pas une

méthode au sens strict du terme, ses processus sont considérés, en pratique, comme des étapes de cycle de vie projet et ses domaines de connaissance dévoilent une linéarité commune aux méthodes axées-plan. Notre deuxième exemple est le *framework* CMMI (Capability Maturity Model Integration). Le choix de ce modèle provient du fait qu'il intègre des processus du domaine d'activité « gestion de projet » qui sont souvent implémentés d'une manière linéaire et lourde, affectant ainsi le cycle de vie projet.

2.2. Exemples des méthodes axées-plan

2.2.1. PMBOK

L'ouvrage de référence en gestion de projet traditionnelle est le PMBOK (PMI, 2008). PMBOK est un corpus de connaissances en gestion de projet qui a été internationalement reconnu (IEEE, norme ANSI) et qui décrit un ensemble de principes et de meilleures pratiques «traditionnelles» de gestion de projets applicables à la majorité des projets. PMBOK considère le projet comme étant un ensemble de processus. Un processus étant une série d'actions exécutées par des personnes et aboutissant à un résultat. Les processus de management de projet (39 en tout) s'attachent à 9 domaines de connaissances et peuvent être classés en cinq catégories : démarrage, planification, réalisation, contrôle et clôture du projet. Chaque discipline décrit les pratiques de la gestion de projet en fonction des processus qui la composent. Ainsi, la discipline de « gestion de la qualité du projet » définira les connaissances et pratiques liées à la planification de la qualité, à l'assurance qualité et au contrôle de la qualité.

PMBOK est devenu le référentiel ultime pour la gestion des projets et ses certifications sont devenues une exigence courante dans le monde anglophone. Cependant, les partisans des nouvelles approches «agiles» reprochent au PMBOK d'être résistant au changement et de produire une documentation excessive (Boehm, 2002). Le PMI (Project Management Institute) rétorque en indiquant que le PMBOK n'est qu'un *framework* ayant pour objectif la satisfaction des clients donnant droit à l'interprétation. Ainsi, de nouveaux efforts sont-ils déployés afin d'arriver à un *mapping* des pratiques agiles dans le contexte du PMBOK, permettant ainsi de bénéficier des forces des deux approches et à pallier leurs faiblesses respectives.

2.2.2. CMMI

La famille CMMI (2006) est un ensemble de normes américaines du SEI (Software Engineering Institute) constituant un *framework* pour l'amélioration des processus de développement logiciel en particulier ceux relatifs au domaine d'activité de gestion de projet. Il est composé de pratiques clés permettant de produire un logiciel de qualité, avec une productivité accrue et dans le respect des budgets et des délais.

CMMI est un modèle qui procède d'une logique d'amélioration progressive sur 5 niveaux de maturité. Il permet d'évaluer la maturité des processus et des pratiques d'une organisation par rapport à ce qu'il faudrait faire idéalement dans les projets. Les 5 niveaux de maturité vont d'un niveau de maturité initial où les projets se déroulent de manière différente selon l'expérience des intervenants, vers la mise en place d'une organisation efficace exploitant les enseignements et accélérant l'optimisation des processus et des outils mis en place dans les précédents niveaux. Chaque niveau du modèle couvre différents domaines d'activité qui sont utilisés dans la réalisation ou le support à la réalisation de produits logiciels. Ces domaines de processus correspondent à un découpage de l'environnement de développement : Gestion des exigences, Planification de projet, *etc.*. Le CMMI cadre les domaines de processus en précisant les objectifs à atteindre et les pratiques correspondantes tels que : Planifier les processus, Fournir les ressources, Assigner les responsabilités, Former les personnes *etc.*.

Même si le modèle CMM/CMMI jouit d'une grande notoriété dans le milieu professionnel, on lui reproche, comme c'est le cas pour toutes les méthodes traditionnelles, d'instaurer une gestion de projet trop lourde et coûteuse. Les organisations ont cherché à alléger les processus du CMMI en essayant d'y intégrer des pratiques émanant des méthodes « légères » dites aussi agiles.

Les méthodes axées-plan focalisent donc sur une planification détaillée, une codification des processus et une rigueur d'implémentation, ce qui caractérise le paradigme classique de la gestion des projets. Dans la section suivante, nous présentons un 2ème paradigme dit « agile » dont le principe est d'introduire l'agilité et la flexibilité à la gestion des projets.

3. Méthodes agiles

3.1. Principes des méthodes agiles

Les méthodes traditionnelles ont perdu un grand nombre de partisans en raison de leur rigidité et leur inadaptabilité au changement (Conboy et Fitzgerald, 2004). Les praticiens et les chercheurs ont alors eu recours à de nouvelles méthodes légères dites Agiles, avec la principale mission de faire face aux changements et donner plus de valeur au client dans l'élaboration du produit. Ces méthodes ont suscité un grand nombre de publications et de débats (Highsmith, 2001). Cependant, les recherches académiques sur le sujet restent limitées, vu que la plupart des publications sont l'œuvre de praticiens ou de consultants (Abrahamsson *et al.*, 2002).

Les méthodes agiles se sont développées séparément, jusqu'en 2001, où leurs créateurs sont arrivés à un consensus sous le label «Agile». Les traits essentiels et communs ont alors été formalisés en un manifeste nommé «le manifeste agile» (Manifeste Agile, 2001) prônant quatre valeurs principales:

- priorité aux personnes et aux interactions par rapport aux procédures et outils,
- priorité aux applications fonctionnelles par rapport à la documentation exhaustive,
- priorité de la collaboration avec le client par rapport à la négociation de contrat,
- priorité de l'acceptation du changement par rapport à la planification.

De ces quatre valeurs, découlent douze principes généraux communs aux méthodes agiles énumérés dans le tableau 1.

Tableau 1. Principes des méthodes agiles

1. Satisfaction du client	7. Auto-organisation
2. Coopération entre le client et les	8. Communication face à face
3. Adaptation au changement	9. Rythme soutenable
4. Livraisons fréquentes	10. Simplicité
5. Fonctionnement de l'application comme premier indicateur d'avancement du projet	11. Excellence technique (Refactorisation)
6. Construction du projet avec des personnes motivées (*Self-empowered team*)	12. Introspection

3.1. Principales méthodes

Il existe une panoplie de méthodes qui incorporent les principes déjà mentionnés. Nous présentons ci-après les méthodes qui ont suscité un grand intérêt à la fois des praticiens et des chercheurs: Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), eXtreme Programming (XP), et Scrum.

3.1.1. ASD

Développée par John Highsmith (2010), cette méthode offre une approche agile et adaptative pour la gestion des projets, avec un taux de changement élevé, en se fondant sur la théorie de la complexité. La planification dans un environnement rapide et imprédictible a beaucoup de risques d'échouer. Ainsi, ASD substitue-t-il le cycle traditionnel de « planifier – concevoir – développer » par le cycle « spéculer – collaborer – apprendre » :

- Spéculer ie. reconnaître l'incertitude régnant sur les projets complexes et encourager l'expérimentation et l'exploration ;
- Collaborer c'est-à-dire travailler conjointement (équipe de développement, clients, consultants externes et fournisseurs de logiciels) pour produire de meilleurs résultats et prendre des décisions pertinentes et à temps ;
- Apprendre en testant les connaissances de l'équipe constamment, à la fin de chaque itération, pour devenir une organisation apprenante (et par extension, agile).

3.1.2. DSDM

DSDM a été créé en Angleterre par un consortium dédié (Abrahamsson P. *et al.,* 2002). La première version de la méthode a vu le jour en 1994. L'idée motrice était de fixer le *timebox* et les ressources en premier, puis d'ajuster le nombre de fonctionnalités à leur assigner. Les techniques clés de cette méthode sont :

- *timeboxing* c'est-à-dire un processus par lequel les objectifs sont réalisés à une date prédéterminée et fixe, de 2 à 6 semaines;
- les règles MoSCoW (*Must, Should, Could, Won't*) classent les fonctions en (i) indispensables (sans quoi le système ne peut fonctionner), (ii) souhaitables (importantes mais pouvant être contournées si les délais sont contraignants), (iii) possibles (facilement contournées) et (iv) éliminées (à classer lors de futurs développements);
- les ateliers Facilités (*facilated workshops*) sont des séances d'échanges organisées avec les utilisateurs pour la collecte d'informations et la prise de décisions.

En 2007, DSDM Atern a vu le jour. Cette nouvelle version de DSDM apporte une définition plus claire du *framework* et un focus sur la communication (DSDM Consortium, 2007).

3.1.3. Scrum

Introduite par Schwaber et Beedle (2002), Scrum est une approche empirique qui applique la théorie de la gestion des processus industriels dans des environnements volatils au développement des système. L'approche est fondée principalement sur la flexibilité, l'adaptabilité et la productivité. L'idée directrice de Scrum est que le développement des systèmes implique plusieurs variables techniques et environnementales (spécifications, délais, ressources et technologie). Ceci augmente la complexité et l'imprédictibilité, d'où le besoin de flexibilité pour remédier immédiatement à tout changement (Abrahamsson *et al.,* 2002). Les principales caractéristiques de la méthode Scrum sont :

- production organisée par une série de *sprints* (itérations) d'une durée de 2 à 4 semaines,
- exigences définies comme des éléments d'une liste appelée « *backlog* du produit »,
- Produit (partiel) conçu, codé et testé pendant le *sprint,*
- équipe choisissant, à partir du *backlog* de produit, les éléments qu'elle s'engage à finir,
- *backlog* de sprint créé collectivement et identifiant les tâches et leur durée estimée (une à 16 heures),
- réunion quotidienne de 15 minutes organisée pour répondre aux questions : Qu'as-tu fait hier ? Que vas-tu faire aujourd'hui ? Et qu'est ce qui ne marche pas ?
- revue de *sprint* organisée à la fin de chaque *sprint*. L'équipe présente les réalisations ainsi qu'une démonstration des nouvelles fonctionnalités ou de l'architecture,
- réflexion régulière sur les bonnes réalisations et les défaillances à la fin de chaque *sprint* (réunion de rétrospective).

3.1.4. eXtreme Programming (XP)

La méthode XP a été mise en œuvre par Kent Beck, Ward Cunningham et Ron Jeffries (1999). Elle est considérée comme une collection de pratiques du génie logiciel venues résoudre les problèmes posés par les longs cycles de développement. Cette méthode tire son nom de son principe de base qui est la réduction du coût des modifications, poussée à l'extrême. Pour atteindre cet objectif, XP s'appuie sur quatre valeurs : simplicité, communication, feedbacks et courage (dans le sens de « sans tabou », si un problème est détecté, il ne doit pas être caché). Ces valeurs (Beck, 1999) sont mises en œuvre au moyen de 12 « pratiques » qui guident le travail de l'équipe XP (Tableau 2).

Tableau 2. Les 12 pratiques XP

1. Jeux de planification	7. *Pair programming* (les programmeurs travaillent en binômes)
2. Livraisons petites et fréquentes	8. Appropriation collective du code
3. Utilisation des métaphores pour l'explication de points techniques, architecturaux, fonctionnels	9. Intégration en continu
4. Simplicité de la conception	10. Rythme durable
5. Tests	11. Client sur site (Un représentant du client est présent pendant toute la durée du projet)
6. Refactorisation fréquente (réusinage fréquent du code)	12. Norme de programmation

3.2. Comparaison entre les méthodes agiles

De nombreuses études comparatives ont été menées pour ordonner, analyser et comprendre le domaine dispersé des méthodes agiles. On s'intéressera principalement aux méthodes présentées dans le paragraphe précédent. Abrahamson *et al.* (2003) ont dressé une comparaison des méthodes selon quatre aspects : gestion des projets, cycle de vie, « principes abstraits versus orientation concrète » et « universellement prédéfinie versus appropriée à la situation ». Les résultats obtenus pour les méthodes ASD, DSDM, Scrum et XP sont résumés en figure 1.

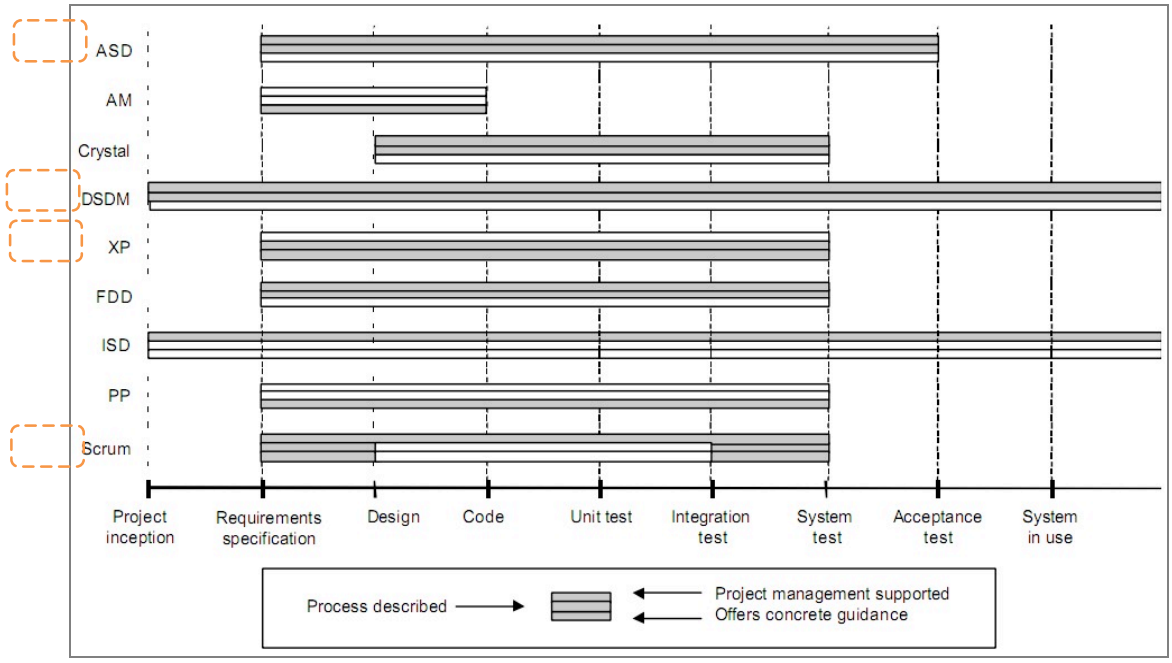

Figure 1. Comparaison des méthodes Agiles. Source : (Abrahamsson *et al.* 2002)

Chacune des méthodes est symbolisée par trois barres. La 1ère barre indique si la méthode fournit un support pour la gestion des projets. La barre centrale indique si un processus du

cycle de vie est couvert par la méthode ; sa longueur indique les phases supportées par la méthode. La 3ème barre indique si la méthode est fondée sur des principes abstraits (couleur blanche) ou si elle fournit une orientation concrète (couleur grise). La couleur blanche fait allusion à l'absence de l'aspect analysé tandis que le gris informe que l'aspect est couvert par la méthode.

3.2.1 La gestion des projets

Les méthodes agiles couvrent cet aspect de manière différente. Même si XP a récemment retenu de nouvelles lignes directrices sur cet aspect, il n'offre pas une vue complète de la gestion du projet. Scrum, par contre, est explicitement destiné à la gestion de projets. Schwabber et Beedle suggèrent d'utiliser d'autres méthodes pour intégrer une approche de développement fondée sur Scrum (par exemple XP). ASD stipule que l'équipe projet doit s'adapter en réponse aux changements survenus dans le projet. DSDM offre un *framework* pour améliorer la capacité de l'organisation à réagir au changement, et contribue à la gestion du projet en facilitant le travail de l'équipe de développement avec un suivi quotidien de la progrèssion du projet.

3.2.2 Le cycle de vie

DSDM essaie de couvrir tout le cycle de développement. ASD couvre la plupart des phases sauf l'initiation du projet, les tests d'acceptation et l'utilisation du système. XP et Scrum se concentrent sur la spécification des besoins, la conception et les tests (jusqu'aux tests système).

3.2.3 Principes abstraits versus orientation concrète

Pour qu'une méthode soit utile et efficace, elle doit apporter une orientation concrète aux équipes de développement à travers des pratiques, des activités et des produits qui décrivent comment une tâche devrait être exécutée. ASD apporte plutôt des concepts et une culture. DSDM ne détaille pas de pratiques, mais il insiste sur le fait que toute organisation devrait développer ses propres pratiques. Dérivée d'un ensemble de *best practices,* XP ne préconise que l'orientation concrète. Scrum définit également un ensemble de pratiques à suivre correspondant aux phases couvertes par la méthode.

3.2.4 Universellement prédéfinie versus appropriée à la situation

Afin de pouvoir réagir au changement, les processus de développement utilisés doivent s'adapter aux situations. Une solution universellement prédéfinie propose une solution prête et non ajustable, ce qui limite visiblement la flexibilité de la méthode et donc sa capacité à réagir aux changements. Les méthodes agiles étudiées dans cet article sont toutes ajustables en fonction de la situation.

Suite à cette présentation des principales méthodes agiles, la section suivante dressera une comparaison entre les paradigmes dominants les méthodes agiles et ceux des méthodes traditionnelles de la gestion de projet.

4. Comparaison entre les méthodes agiles et traditionnelles

Les méthodes traditionnelles et les méthodes agiles se sont développées dans deux milieux diamétralement opposés. Les premières sont venues introduire de la discipline dans les grands projets gouvernementaux ayant des coûts d'échec colossaux. Les méthodes agiles, quant à elles, se sont développées dans la « bulle » du dotcom sur des petits projets web avec des coûts d'échec frôlant le zéro. Les méthodes agiles ont introduit un nouveau paradigme pour

pallier les défaillances des méthodes traditionnelles et garantir un meilleur taux de succès des projets.

Cependant, des études empiriques ont démontré que le paradigme agile n'est pas approprié à toutes les situations et que les approches traditionnelles offrent dans certains cas de meilleurs résultats. En effet, l'une des plus grandes limitations des méthodes agiles est la grande taille des équipes. Cockburn et HighSmith (2001*)* concluent qu'avec l'augmentation de la taille, la coordination entre les interfaces devient un problème crucial. L. Constantine (2001) et M. Fowler (2001) indiquent que la communication face-à-face prônée par les méthodes agiles devient difficile et complexe si l'équipe de développement dépasse les 20 personnes. En revanche, les méthodes traditionnelles axées-plan sont mieux appropriées aux projets à grande échelle.

Une autre limitation considérable des méthodes agiles consiste en la gestion des projets critiques. Les mécanismes de contrôle de qualité incorporés dans les méthodes agiles ne prouvent pas que les produits développés sont sans danger. Les spécifications formelles, les tests rigoureux et d'autres techniques d'analyse et d'évaluation inclus dans les méthodes traditionnelles offrent de meilleurs mécanismes, même si ils sont plus lourds, pour gérer les projets critiques (Turk *et al.*, 2002). Le tableau 3 résume les principales différences entre ces deux approches.

Tableau 2. Différences entre les méthodes agiles et traditionnelles

Critères	Méthodes traditionnelles	Méthodes agiles
Approche	prédictive	adaptative
Mesure du succès	conformité au plan	valeur client
Domaine	prévisible	imprévisible/exploratoire
Retour sur investissement	à la fin du projet	au début du projet
Style de management	autocratique	décentralisé
Culture	commande, contrôle	*leadership*, collaboration
Environnement	stable, faible taux de changement	turbulent, taux élevé de changement
Relation client	interaction au besoin, insistance sur les exigences du contrat	un représentant du client est dévoué au projet, insistance sur la priorisation des incréments
perspective de changement	durabilité de changement	adaptabilité au changement
Focus	processus	humain
Communication	explicite	tacite
Documentation	lourde	faible
Cycles	en nombre limité	nombreux
Planification en amont	complète	minimale
Taille du projet	grande taille	petite taille
Taille de l'équipe	grande taille	petite taille/créative
Développement	conception extensive, longs incréments	conception simple, courts incréments

On peut supposer alors que les méthodes agiles et traditionnelles ne sont pas incompatibles et qu'elles pourraient être combinées en cas de besoin. Les méthodes formelles pourraient être utilisées d'une manière agile pour traiter les parties critiques du système et offrir une

meilleure qualité et un taux tolérable de confiance. La section suivante présentera quelques méthodes dites hybrides qui proposent de profiter des avantages et d'esquiver les inconvénients des deux catégories de méthodes.

5. Méthodes Hybrides

Boehm et Turner (2003) affirment que les méthodes agiles, comme les méthodes traditionnelles, ont des points forts et des points faibles. Il est donc important pour les organisations d'équilibrer les deux approches pour bénéficier de leurs forces et pallier leurs faiblesses. La littérature actuelle n'offre cependant que quelques approches hybrides pour l'amélioration du développement logiciel. Dans le prochain paragraphe, nous présentons trois d'entre elles, à savoir le modèle de Boehm et Turner, le modèle de Sidky et l'intégration du CMMI et des pratiques agiles.

5.1. Le modèle de Boehm et Turner

L'un des premier modèle à adopter les pratiques agiles est le modèle Boehm et Turner (2003). Ce modèle identifie les risques relatifs au projet et d'indique la méthode à adopter (disciplinée, agile ou une combinaison des deux). Le modèle consiste en cinq étapes clés : Analyse du risque, Comparaison du risque, Analyse de l'architecture, Ajustement du cycle de vie et Exécution et *monitoring*. Boehm et Turner ont identifié cinq facteurs d'agilité représentés par des axes d'analyse (figure 2) qui interviennent dans la sélection des méthodes agiles ou traditionnelles :

- la taille c'est-à-dire le nombre des membres de l'équipe,
- la criticité du projet,
- le dynamisme qui traduit le degré de changement des spécifications,
- le personnel ou degré de compétence et d'expérience de l'équipe,
- la culture de l'organisation.

Chaque axe représente une dimension d'agilité. Quand les données relatives au projet sont projetées sur les axes, les différents points sont reliés et les formes obtenues sont analysées. Si la forme est autour du centre, le modèle suggère l'utilisation d'une méthode agile. Une forme qui tend vers les péripéties indique l'usage d'une méthode axée-plan. Toute autre forme suggère un usage combinant les méthodes axées-plan et les pratiques agiles.

Le modèle peut certainement être utilisé comme un mécanisme pour commencer l'évaluation de l'agilité et la sélection d'un projet pilote agile. Cependant, il ne présente pas de détails quant à l'application des pratiques agiles. Le défi majeur des organisations réside dans l'adaptation des pratiques agiles à leur contexte de développement et dans l'application des pratiques les plus appropriées dans le cadre des activités de l'organisation.

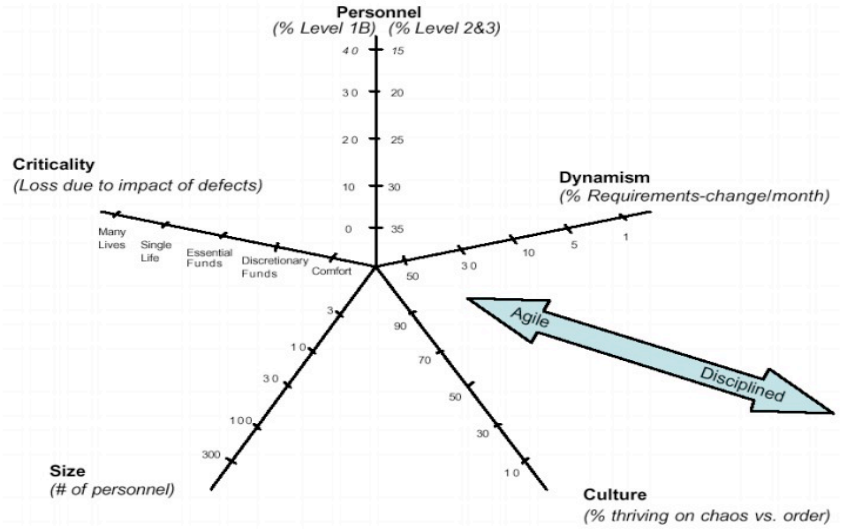

Figure 2. Les 5 facteurs d'agilité de Boehm & Turner. Source : Boehm and Turner (2003).

5.2. Le modèle de Sidky

Ahmed Sidky (2007) présente un *framework* pour l'intégration des pratiques agiles dans le processus de développement traditionnel des projets. Ce *framework* expose des niveaux pour l'évaluation de l'agilité. Comme c'est le cas pour Boehm et Turner, Sidky insiste sur la nécessité d'évaluer le besoin d'une organisation en agile en premier, avant de prendre la décision d'adopter des pratiques agiles. Ceci permet à l'organisation de n'adopter que les pratiques agiles qu'elle a la capacité de concrétiser. Sidky fournit alors un *framework* pour les projets ainsi qu'une évaluation de l'agilité de l'organisation, en quatre étapes :

- identification des facteurs de discontinuité. Avant l'évaluation, l'organisation définit sa capacité d'adoption de l'agilité. Les facteurs de discontinuité sont les facteurs qui peuvent prévenir l'organisation de prendre la décision de se lancer dans l'agile. L'absence du soutien de la direction ou un budget insuffisant sont des facteurs de discontinuité ;
- Évaluation du niveau de l'organisation par identification du plus haut niveau d'agilité que le projet pourrait atteindre ;
- Évaluation de l'inclination de l'organisation à supporter le niveau d'agilité souhaité ;
- Réconciliation en ressortir les pratiques que l'organisation veut et peut adopter.

L'agilité est évaluée sur une échelle à cinq niveaux : collaboratif, évolutif, efficace, adaptatif et environnant. Cette approche est utilisée dans le modèle IDEAL qui est un modèle d'amélioration des processus logiciels fondé sur SW-CMM. Il est appropriée aux situations où les organisations ont recours à l'évaluation fondée sur CMMI. Chaque niveau d'agilité se caractérise par des pratiques agiles particulières.

Le *framework* de Sidky ouvre d'intéressants horizons pour l'adoption de l'agile, mais il reste à prouver sur le plan empirique. Il ne démontre pas pourquoi l'adoption de l'agile doit suivre l'ordre spécifié dans le *framework*. Le *framework* affirme être fondé sur CMM, cependant, il n'offre cependant pas de détails sur les points suivants :

- correspondance des pratiques agiles avec CMMI,
- diminution de l'effort en évaluation pour l'adoption des pratiques agiles,
- validation de l'impact des améliorations sur le processus de développement agile (Pikkarainen, 2008).

5.3. Intégration du CMMI et des pratiques agiles

D'après le SEI, institut ayant développé le CMMI, les pratiques agiles devraient être compatibles avec CMMI, vu que ce dernier représente un cadre pour l'auto-amélioration. L'intégration des pratiques agiles avec celles des méthodes axées-plan a toujours semblé impossible aux praticiens. Ceci est principalement dû aux idées reçues sur l'intégration de l'agile et du CMMI (Glazer *et al.,* 2008). Cependant, plusieurs recherches ont essayé d'intégrer les pratiques agiles dans les processus CMMI. Les stratégies suivies pour cette intégration peuvent être classées comme suit (Alkhatib, 2004) : *front-end*, *back-end*, concurrente, *re-engineering*.

L'approche *front-end* suggère l'usage des pratiques agiles avant l'implémentation de l'approche CMMI dans le système (Boehm, 2002), contrairement à l'approche *back-end* qui stipule qu'une fois le système est développé selon l'approche CMMI, les pratiques agiles pourront être utilisées pour exercer les activités de maintenance.

La stratégie concurrente préconise une implémentation en parallèle des pratiques agiles et des processus CMMI. Un des problèmes majeur de cette approche est l'affectation des ressources humaines aux différents modules/composants d'un même projet. Les méthodes agiles requièrent des expériences et des rôles différents de ceux des méthodes fondées sur CMM. Pour contourner ce problème, la stratégie concurrente identifie certains modules d'un projet ou des projets en entier, et puis construit en conséquence les équipes qui utiliseront l'agile ou CMMI.

La stratégie de *Re-engineering* requiert beaucoup de temps et d'efforts et insiste sur la nécessité de repenser les pratiques du CMMI et des méthodes agiles. Par exemple, CMMI devrait insister sur la collaboration et sur la vitesse couplée avec la qualité ; les méthodes agiles devraient générer de la documentation au-delà du code, *etc.*

Suite à cette présentation des méthodes traditionnelles, agiles et en fin hybrides, la section suivante est consacrée au choix de la méthode de gestion la plus appropriée pour un contexte de projet donné.

6. Choix d'une méthode

Les méthodes agiles promettent, contrairement aux méthodes traditionnelles, de garantir une meilleure satisfaction client, un taux inférieur de défauts, des livraisons rapides et une solution aux changements des spécifications client (Boehm and Turner, 2003). Cependant, ces méthodes ne sont appropriées qu'aux petits projets à faible complexité. Les méthodes hybrides sont alors apparues pour relever le défi d'équilibrer les deux approches, agiles et traditionnelles. R. Glass (2002) confirme qu'il n'existe pas une unique méthode pour la construction des logiciels, mais que les méthodes devraient être choisies selon les problèmes posés. Des problèmes différents requièrent des méthodes différentes. Cependant, les recherches effectuées dans ce domaine restent insuffisantes (Glass 2004). Glass suggère en effet que de nouvelles recherches soient effectuées ayant pour objectifs de :

- établir une taxonomie des méthodes disponibles,
- établir une taxonomie du *spectrum* des domaines des problèmes,
- effectuer un *matching* entre les deux taxonomies.

Glass ne nie pas les difficultés que revêtent les deux premières étapes mais affirme que praticiens et chercheurs doivent se pencher ensemble sur la classification des méthodes et des problèmes. La même observation a été introduite par Dyba et Dingsoyr (2008) qui conclurent qu'il y a un grand besoin en recherches pouvant déterminer les situations où les conseils des praticiens peuvent être convenablement implémentés et réduire ainsi l'écart entre la recherche et l'industrie du logiciel. En effet, les praticiens organisent fréquemment des débats pour essayer de résoudre la question : agile ou traditionnel ? La décision d'adopter une méthode de développement n'est pas facile, d'autant plus qu'en théorie, comme en pratique, le focus s'est fait sur la description des méthodes et non sur les facteurs influençant le choix de ces méthodes (Dyba et Dingsoyr, 2008). La recherche doit donc répondre au problème de l'optimalité du choix de la méthode de développement en considérant entre autres le projet, l'équipe et l'organisation.

Parmi les rares travaux qui ont pour objectif d'étudier les facteurs influençant le choix d'une méthode, citons A. Cockburn (2000) qui identifie trois facteurs : la taille de l'équipe, la criticité du système et les particularités du créateur de la méthode (partant du fait que la méthode est une prévention contre les craintes de son créateur).

Les deux premiers facteurs semblent faire unanimité chez les chercheurs. Glass (2001) quant à lui, ajoute à ces deux facteurs, le domaine d'application (scientifique ou d'ingénierie) et le degré d'innovation que nécessite le projet.

Le modèle de Boehm et Turner (2003) suggère une analyse des projets selon cinq axes : taille de l'équipe et du projet, criticité, personnel, culture de l'organisation (être à l'aise dans l'ordre ou dans le chaos) et dynamisme (pourcentage du changement des spécifications par mois).

Dans une étude plus globale menée par J. McAvoy et D. Sammon (2005), onze facteurs ont été identifiés et regroupés en quatre catégories : projet, équipe, client et organisation (tableau 3).

Tableau 3. Facteurs critiques de McAvoy & Sammon (2005)

Catégorie	Facteurs critiques d'adoption
Projet	- Durée du projet - Acceptation du changement - Criticité du projet
Équipe	- Taille de l'équipe - Compétences des membres de l'équipe
Client	- Localisation du client - Implication du client
Organisation	- Structure du système d'organisation et de *reporting* - Processus - Exigence en documentation - Aménagement des lieux du travail

Le défi que pose le choix d'une méthode est son caractère longitudinal (McAvoy et Butler, 2009). En effet, cette décision inclut de multiples autres décisions prises sur une longue période. Il faudrait donc évaluer constamment la pertinence du choix de la méthode et affiner ainsi les mesures des facteurs, au fur et à mesure de l'exécution des projets.

7. Conclusion

Il va sans dire que les méthodes agiles ont introduit un nouveau paradigme dans la gestion des projets informatiques. Venues contrer une approche axée-plan, lourde, mise en oeuvre pour gérer initialement les grands projets gouvernementaux, ces méthodes ont gagné de plus en plus de partisans au fil des années. Leur caractère léger, flexible et adaptatif aux changements ainsi que leur impact positif et indéniable sur la productivité des équipes, leur ont permis de s'imposer dans le milieu professionnel (Laanti *et al.,* 2011).

Cependant, l'approche agile présente des lacunes que seul un appariement avec les méthodes traditionnelles pourrait combler. Ainsi, sont nées les méthodes hybrides qui tentent de « réconcilier » l'agile et le traditionnel, en intégrant un ensemble de pratiques agiles dans les processus de développement traditionnels. Les travaux récents de recherche se sont orientés vers le *mapping* entre les processus formels (émanant des standards tels que CMMI ou PMBOK) et les pratiques agiles. Cette combinaison s'est révélée une approche viable pour une meilleure qualité et productivité (Magdalenoa *et al.,* 2012). Il est pourtant important d'évaluer la disposition d'une organisation ou d'un projet à accepter l'agilité avant de pouvoir adopter ces pratiques. L'évaluation de l'amélioration des processus suite à l'adoption de l'agile est un deuxième volet qui doit être impérativement traité. Les méthodes formelles ayant parcouru un long chemin dans la voie de l'évaluation, peuvent être utilisées comme base d'un modèle d'évaluation de l'amélioration des processus de développement. Il est pourtant primordial de garder en perspective l'aspect flexible que viennent introduire les pratiques agiles et d'alléger alors ces modèles d'amélioration. Ainsi, la mise en place d'un *framework* d'assistance et d'évaluation pour le choix et l'adoption d'une méthode (agile ou hybride), fondé sur un modèle traditionnel (CMMI ou PMBOK) semble constituer une problématique qui devrait faire l'objet de recherches dans l'avenir.

Références

Royce W. (1970). Managing the Development of Large Software Systems. *IEEE WESCON Conference*.

Boehm B. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, Vol. 21, No. 5, 61-72.

PMI. (2008). *Guide to the project management Body of knowledge*. (4h Edition). Pennsylvania : PMI Inc.

Boehm B. (2002). Get Ready For The Agile Methods, With Care. *IEEE Computer*, Vol. 35, No. 1, 64-69.

CMMI Product Team. (2006) *CMMI® for Development*. (Version 1.2). Pittsburgh: Carnegie Mellon University.

Conboy K., Fitzgerald B. (2004). Toward a conceptual framework of agile methods: A study of agility in different disciplines. *Proceedings of the ACM Workshop on Interdisciplinary Software Engineering Research, Newport Beach, CA, 2004. 37–44.*

Highsmith J. (2001) Opening Statement. *Cutter IT Journal, The Great Methodologies Debate: Part 1*. Décembre 2001, Vol. 14, No. 12

Abrahamsson P., Salo O., Ronkainen J., Warsta J. (2002) *Agile software development methods – Review And Analysis.* VTT Publication ESPOO. Julkaisija – Utgivare

Manifeste Agile (2001) www.agilemanifesto.org

Highsmith J. (2000). Retiring lifecycle dinosaurs. *Software Testing & Quality Engineering (STQE) magazine.* 2, 4 Juillet/Août, 22–28.

DSDM Consortium (2007). DSDM Atern, www.dsdm.org/atern/

Beck K. (1999). Embracing change with Extreme Programming. *IEEE Computer*, Issue 10 October 1999, Vol. 32, 70-77.

Abrahamsson P., Warsta J., Siponen M.T., Ronkainen J. (2003) New directions on Agile Methods: A comparative Analysis. Proceedings de la *IEEE International conference on Software Engineering*. Portland: 2003.

Cockburn A. and Highsmith J. (2001*),* Agile Software Development: The People Factor. *Computer*, Novembre 2001, 131-133.

Constantine L. (2001), Methodological Agility. *Software Development*, Juin 2001, 67-69.

Fowler M. (2001), *Is Design Dead ? Extreme Programming Explained*. Boston : Addison Wesley Longman.

Turk R., France R., Rumpe B. (2002). Limitations of Agile Software Processes. *Proceedings de la 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering—XP2002, Alghero, 2002.*

Boehm B., Turner R. (2003). Observation on Balancing discipline and agility. *Proceedings of Agile Development Conference, Salt Lake City, 2003.*

Sidky A., (2007). *A Structured Approach to Adopting Agile Practices: The Agile Adoption Framework.* Mémoire de doctorat, Virginia Tech.

Pikkarainen M. (2008). Towards a Framework for Improving Software Development Process Mediated with CMMI Goals and Agile Practices. *VTT Publications 695*.

Glazer H., Dalton J., Anderson D., Konrad M., Shrum S., (2008) CMMI® or Agile: Why Not Embrace Both! http://www.sei.cmu.edu/publications/

Alkhatib G I. (2004) Agile methods – CMMI-SW: linking the two extremes, *Proceedings of the Second Conference on Administrative Sciences*, Dhahran, Avril 2004, 461-472.

Glass R.L. (2002). Searching for the Holy Grail of software engineering. *Communication the ACM*, Mai 2002, Vol. 45, No. 5.

Glass R.L. (2004) Matching Methodology to Problem domain. *Communication of the ACM* Mai 2004, Vol. 47, No. 5

Dyba T., Dingsoyr T. (2008) Empirical studies of agile software development: A systematic review. *Inform. Softw. Technol*., vol. 50, no. 9-10, 833–859.

McAvoy J., Sammon D. (2005) Agile methodology adoption decisions: An innovative approach to teaching and learning. *Journal of information and systems Education 2005 Vol. 16(4)*.

Cockburn A. (2000). Selecting a project's methodology. *IEEE Software* Juillet/Août 2000, 64-71.

Glass R. L. (2001). Agile Versus Traditional: Make Love, Not War! *Cutter IT Journal* Décembre 2001. V. 14, N. 12: 12–18.

Boehm B., Turner R. (2003). *Balancing Agility and discipline: A guide for the perplexed*, Boston: Addisson-Wesley.

McAvoy J., Butler T. (2009) The role of project management in ineffective decision making within Agile Software development projects. *European Journal of Information Systems.* V. 18, N 4, Août 2009.

Laanti M., Salo O., Abrahamsson P. (2011) Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *The Journal of Systems and Software.* V. 53. N.3 : 276–290

Magdalenoa A. M., Wernera C. M. L., Mendes de Araujob R. (2012) Reconciling software development models: A quasi-systematic review. *The Journal of Systems and Software.* V. 85, N. 2: 351-369