Proposition d'une méthodologie agile en intelligence d'affaires pour réduire les risques d'échecs

par

Étienne Rivard

Essai présenté au CeFTI
en vue de l'obtention du grade de maître en technologies de l'information
(maîtrise en génie logiciel incluant un cheminement de type cours en technologies de l'information)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, septembre 2012

Sommaire

Les projets d'entrepôt de données, comme les autres types de projets en technologies de l'information, ont un faible taux de succès.[9] Selon plusieurs auteurs, ce taux est attribué principalement à un manque de qualité dans les données des entrepôts et à la difficulté qu'ont les équipes TI de comprendre le client et ses besoins. [33] [42] [46]

Les équipes TI ont eu du succès avec les méthodologies agiles lorsqu'appliquées au développement logiciel. Cet essai émet l'hypothèse que l'application des principes agiles dans un contexte d'entrepôt de données est susceptible de réduire les risques d'échec.

L'essai propose une méthodologie, nommée MAIA (Méthodologie Agile en Intelligence d'Affaires). Celle-ci est composée de pratiques venant de méthodologies telles que XP et *Scrum.* Ces pratiques ont été étudiées, testées et prouvées efficaces pour améliorer le taux de succès d'un projet TI.

La méthodologie MAIA apporte aussi des pratiques des méthodologies traditionnelles à l'intelligence d'affaires pour compléter les pratiques agiles afin de combler les besoins particuliers des entrepôts de données.

Remerciements

Il y a six ans, je voulais faire un retour aux études. Mon épouse, Karine Brière, a eu l'idée de génie de me proposer d'entreprendre ma maitrise. Elle était tellement convaincue que j'avais les capacités d'entreprendre cette grande aventure que j'ai décidé d'aller de l'avant.

Tout au long de ces années, Karine a été un support moral extraordinaire. Je la remercie pour ce temps où elle s'est occupée seule de notre fils pendant mes cours, mes devoirs et l'écriture de cet essai. Elle a su me montrer qu'il faut se concentrer sur ce qui est le plus important et ne jamais lâcher. C'est aussi sa rigueur en français qui a contribué au professionnalisme de cet essai.

Elliott Rivard, mon garçon, est une source d'inspiration. Depuis qu'il est tout petit, il a démontré une soif de savoir et une curiosité contagieuse. Je me suis inspiré de sa ténacité à creuser un sujet pour le maîtriser dans ma propre recherche pour l'écriture de cet essai. Je suis heureux qu'Elliott soit dans ma vie.

Avec mes besoins affectifs amplement comblés, j'ai demandé de l'aide économique à deux personnes clés dans mon épanouissement professionnel, Oumar Ba et Pierre Trahan de Cascades ULC. Oumar et Pierre ont tout de suite cru en moi ainsi qu'en mon projet. Ils m'ont aidé en m'offrant des ressources humaines et matérielles pour l'accomplissement de celui-ci. Merci Oumar et Pierre de m'avoir fait confiance jusqu'au bout!

Deux autres collègues de Cascades m'ont aidé à la fin de cette aventure et j'en profite pour aussi remercier Jessy Abran et Luc Pothier. Ils m'ont donné le temps que j'avais besoin pour assister à mes cours et à rédiger cet essai.

Je souhaite aussi remercier mes parents pour avoir toujours été là pour moi.

Enfin, merci à mes directeurs, Patrice Roy et Laura Francheri, sans qui cet essai ne serait pas de cette qualité.

Table des matières

Liste des tableaux

Liste des figures

Glossaire

Magasin de données	Sous-ensemble logique d'un entrepôt de données complet.
Métadonnée	Toute information dans un entrepôt de donnée qui n'est pas la donnée en tant que telle, par exemple les structures des tables de l'entrepôt, les statistiques d'utilisation et d'exécution des ETC et les règles de transformation des données.
Tableau de bord	Outil graphique affichant les indicateurs clés de performance pour un domaine donné, destiné aux dirigeants d'entreprise.
Test d'acceptation	Test fait par l'utilisateur pour déterminer si le produit lui est acceptable, s'il respecte ses demandes.
Test fonctionnel	Test d'un ensemble d'entités pour évaluer leur bon fonctionnement global, incluant les interactions.
Test de régression	Test qui assure que le logiciel conserve toutes les fonctionnalités existantes lors d'ajout de nouvelles fonctionnalités.
Test unitaire	Test granulaire d'une méthode spécifique ou une seule entité, normalement isolé des autres entités.

Liste des sigles, des symboles et des acronymes

DSDM	*Dynamic Software Development Methodology* : Méthodologie de développement logiciel
ETC	Processus utilisé dans l'entreposage de données afin d'extraire (E) les données d'un système opérationnel, les transformer (T) et les charger (C) dans un entrepôt
OLAP	On-Line Analytical Processing, technique informatique d'analyse multidimensionnelle, qui permet aux décideurs, en entreprise, d'avoir accès rapidement et de manière interactive à une information pertinente présentée sous des angles divers et multiples, selon leurs besoins particuliers
RAD	*Rapid Application Development* : Méthodologie de développement logiciel
TDD	*Test-Driven Developmen t*: Méthodologie de développement logiciel
XP	*eXtreme Programming* : Méthodologie de développement logiciel

Introduction

Il fut un temps où les projets en TI étaient perçus comme des dépenses dans l'industrie, au même titre que l'électricité ou les fournitures de bureau. Cette notion a clairement été illustrée par le président d'une entreprise papetière il y a quelques années quand il affirma : « Ici on fait du papier, pas de l'informatique! L'informatique ce n'est qu'une dépense. »

Les clients internes aux équipes TI de ces entreprises n'avaient qu'une exigence : que leurs besoins en informatique soient comblés. Dans ce contexte, le respect d'un budget ou d'un échéancier n'était pas un enjeu majeur. Cependant, ces mêmes clients ont commencé à s'apercevoir que les coûts engendrés par leurs départements des TI étaient majeurs, et que ces départements étaient les seuls à ne pas gérer leurs projets comme les autres départements, tels que l'ingénierie ou le développement de produit par exemple. Ce temps est depuis longtemps révolu et les projets TI sont maintenant examinés, mesurés et critiqués en fonction d'un budget pré-établi. Le faible taux de succès de ces projets a amené les TI à concevoir des méthodes de travail pour atteindre le but ultime dans la gestion de projet : terminer le projet dans le temps prévu, en respectant les coûts établis dans le budget et en livrant les fonctionnalités promises.

Depuis ce temps, des méthodologies formelles, empruntées du monde de l'ingénierie, ont été appliquées aux TI pour la gestion de projet. Comme toute méthodologie, elles ont eu leur lot de succès et d'échecs.

Plus récemment, un mouvement dans la communauté TI a amené une nouvelle façon d'exécuter les projets TI. Plusieurs méthodologies ont vu le jour pour réduire le taux d'échec des projets. Plusieurs d'entre elles, dont celles qui seront discutées dans cet essai, sont reconnues comme étant des méthodologies agiles. La littérature démontre que ces méthodologies ont effectivement un effet sur le succès de projets TI.[14] [29]

Ces méthodologies ciblent majoritairement les projets TI qui concernent le développement logiciel, cependant d'autres types de projets TI pourraient profiter de celles-ci.

Le type de projet TI examiné dans cet essai est l'entreposage de données. Un projet d'entrepôt de données met en place un environnement pour l'analyse de données d'affaires. Un tel projet, comme les autres types de projets TI, comporte des risques d'échec et un taux de succès faible. Ceci suggère la question suivante : est-ce que les méthodologies agiles peuvent contribuer au succès d'un projet d'entrepôt?

Pour répondre à cette question, l'essai est divisé en trois chapitres. Le premier chapitre décrit les méthodologies traditionnelles, de la méthodologie en cascade à celles d'auteurs reconnus dans le monde de l'intelligence d'affaires. Ensuite, une revue des raisons majeures d'échec dans les projets d'entrepôt est faite, basée sur la littérature scientifique.

Le deuxième chapitre montre quelques méthodologies agiles utilisées dans le monde du développement logiciel. Ce chapitre comprend une description de l'origine de chaque méthodologie et une présentation des aspects majeurs de celles-ci.

Le troisième chapitre propose une méthodologie orientée vers les projets d'entrepôts de données, basée sur les méthodologies agiles. Le cycle de vie de la méthodologie est calqué sur les méthodologies agiles existantes et emprunte leurs meilleures pratiques. La méthodologie apporte aussi des points spécifiques au monde de l'intelligence d'affaires qui ne sont pas abordés par les méthodologies agiles. En se basant sur la littérature scientifique, l'essai justifie les choix de pratiques en fonction de leur apport à la réduction des risques dans un projet TI.

Chapitre 1
Méthodologies traditionnelles pour les projets d'entrepôt

Ce premier chapitre présente trois méthodologies : la méthodologie traditionnelle de développement de logiciel en cascade ainsi que les deux méthodologies les plus reconnues de conception d'entrepôts. Un rapprochement est fait par la suite entre ces méthodologies. Enfin, le reste du chapitre expose les problématiques vécues dans les projets d'entrepôt.

1.1. Entrepôts de données

Dans une entreprise typique, l'information peut se trouver sous deux formes, soit les données opérationnelles et l'entrepôt. Majoritairement, les données opérationnelles sont créées par les systèmes opérationnels, par exemple un progiciel de gestion intégré, alors que l'entrepôt consomme ces données opérationnelles pour permettre une analyse des opérations de l'entreprise.

Les données opérationnelles représentent principalement les processus de l'entreprise. Par exemple, dans une entreprise manufacturière, ce sont entre autres les commandes des clients, les produits, les matières premières, les bons de matériel, les factures et bien sûr, les données financières. C'est en fait toute donnée permettant à l'entreprise de fonctionner au jour le jour. Les données opérationnelles changent, pour refléter l'état des opérations.

L'entrepôt, en contraste, est une vue historique de l'entreprise et de ses opérations. On y retrouve des informations telles que les commandes, mais qui peuvent être mises en contexte avec le bon d'expédition, la facture et le paiement du client pour établir une vision complète du processus de commande à l'encaissement pour fin d'analyse. Le focus n'est souvent pas la commande unitaire, mais les ensembles qui peuvent démontrer des tendances significatives.

1.2. Méthodologie en cascade

Traditionnellement, les projets en technologies de l'information, l'ensemble des matériels, logiciels et services utilisés pour la collecte, le traitement et la transmission de l'information, [28] étaient réalisés à l'aide d'une méthodologie empruntée au monde de la construction. Ce processus commence par la définition des besoins du client, pour ensuite créer un plan et construire un bâtiment.

La méthodologie en cascade stipule que chaque étape doit se terminer avant de débuter l'étape suivante. Par exemple, la phase d'analyse détaillée doit être complétée avant de commencer la réalisation.

La figure 1.1 illustre les différentes étapes d'un projet en cascade. La première étape consiste à préciser les besoins systèmes. Il faut avoir une idée claire des besoins utilisateurs pour concevoir une architecture complète du système à créer. La deuxième étape est la définition des besoins détaillés du client pour le logiciel à créer. Tous les besoins sont définis et négociés à cette étape. Une fois cette étape complétée, tout changement est catalogué et négocié dans un processus de demande de changement. La troisième étape est l'analyse détaillée. L'ensemble du logiciel est pensé et conçu avant qu'une seule ligne de code ne soit écrite. La quatrième étape est la création du logiciel. La tâche du programmeur se limite à interpréter les documents d'analyse et à les transformer en code. La cinquième étape consiste à tester l'ensemble du logiciel. Le logiciel est complet à cette étape et est testé entièrement par une équipe d'assurance qualité. Enfin, le logiciel est mis en opération.

Les avantages d'utiliser cette approche sont d'imposer une discipline aux développeurs de logiciels et de permettre de déterminer facilement la progression du projet. En effet, les phases du projet ont un début et une fin bien définis et rendent possible la documentation des jalons. [40]

La rigidité du passage d'une étape à l'autre, sans permettre le retour à une phase subséquente, vient du monde de la conception matérielle, où les coûts de rebrousser chemin au milieu d'un projet sont majeurs. [44]

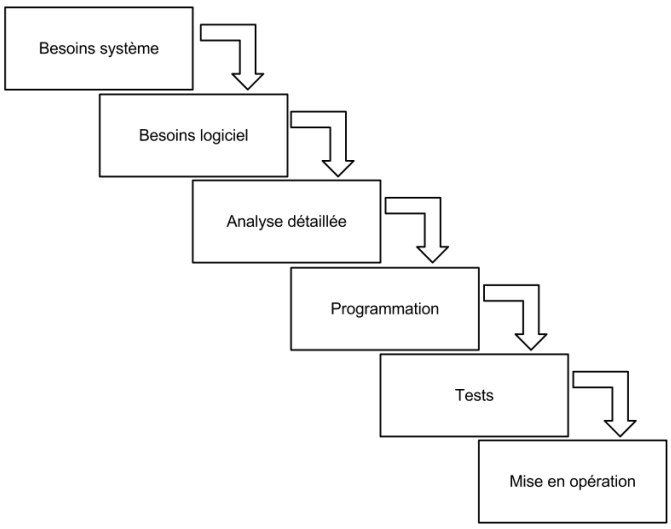

Figure 1.1 Étapes d'un projet en cascade

1.3. Méthodologie d'Inmon

Bill Inmon est un informaticien américain considéré comme le père des entrepôts et écrivain d'un livre [18] expliquant une approche pour les créer. Dans son approche, il faut d'abord créer l'entrepôt qui comprend l'ensemble des données de l'entreprise, pour ensuite alimenter de mini-entrepôts de données départementales. Chaque mini-entrepôt vise un secteur particulier de l'entreprise et sera la source des requêtes et rapports utilisés par les décideurs de l'entreprise.

Inmon divise l'environnement de bases de données de l'entreprise en quatre niveaux :

- opérationnel,

- entrepôt de données atomique,
- départemental et
- individuel.

Les trois derniers niveaux constituent l'entrepôt. Le premier niveau comprend les bases de données de logiciels patrimoniaux et autres systèmes de traitement transactionnel. Les données sont transformées entre le premier et le deuxième niveau. Pour transformer les données, un processus appelé ETC (pour extraction, transformation et chargement) est utilisé. Il s'agit d'un ensemble de processus qui extraient les données des systèmes sources, qui transforment les données et les chargent dans l'entrepôt cible. Avec cette approche, les données des départements sont toujours cohérentes, car elles proviennent toutes du même entrepôt.

Développer suivant la méthodologie d'Inmon demande une connaissance approfondie de l'entreprise. Inmon recommande l'utilisation de modèles de données déjà existants sur le marché pour le domaine d'affaires de l'entreprise. [6] La modélisation de la base de données repose sur les techniques traditionnelles d'entités-association et de normalisation. Le modèle d'entités-association est un modèle de données conceptuel de haut niveau contenant des entités et les associations entre elles, traditionnellement utilisé pour concevoir des bases de données en informatique.

La méthodologie d'Inmon est destinée à un public composé de professionnels des technologies de l'information. Les utilisateurs jouent un rôle passif dans le développement de l'entrepôt; [6] ils sont cependant plus impliqués au moment d'illustrer leurs besoins.

Inmon décrit sa méthodologie en la comparant au *Software Development Life Cycle* (SDLC). Le SDLC est alimenté par les besoins des utilisateurs. Inmon prétend que sa méthodologie est exactement le contraire et l'appelle CLDS (en inversant les lettres SDLC). La figure 1.2 illustre le CLDS. Le CLDS débute avec les données. Elles sont intégrées et testées puis une analyse est faite pour découvrir des tendances. Ensuite, des programmes sont créés pour

alimenter le système de support à la décision. Ultimement, une analyse des résultats des programmes est faite pour enfin comprendre les besoins. [18] Le cycle se répète pour chaque ensemble de données disponible.

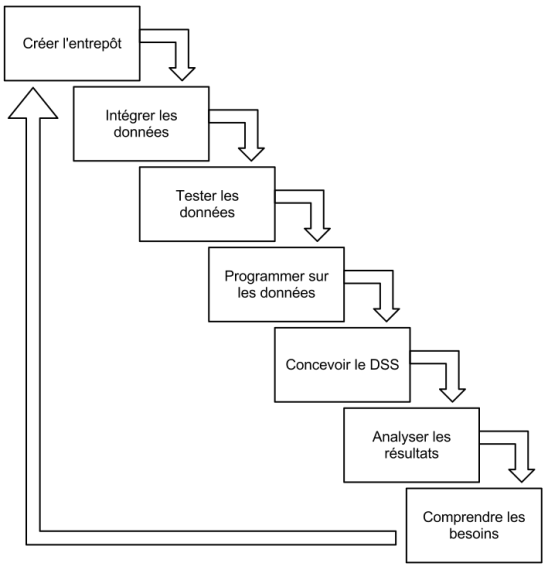

Figure 1.2 Étapes d'un projet avec la méthodologie d'Inmon

1.4. Méthodologie de Kimball

Ralph Kimball prend une approche tout à fait différente d'Inmon pour la création d'entrepôts. Premièrement, il a développé une technique pour modéliser un entrepôt qui, à l'époque, était révolutionnaire : le modèle dimensionnel. Au lieu d'être basé sur le modèle entités--association, le modèle dimensionnel repose sur des tables représentant des faits et des dimensions. Le modèle dimensionnel peut être représenté dans la plupart des cas par un modèle entités-association, mais est plus simple à comprendre dans sa forme dimensionnelle. Les faits sont typiquement des évènements mesurables qui se produisent dans l'entreprise. Il est simple de mesurer la quantité de produit commandée, le montant d'une facture ou le nombre de jours de retard d'un paiement. Les dimensions sont des descriptions textuelles

d'élément de l'entreprise. [22] Les dimensions permettent d'analyser les faits sous différents angles, par exemple des ventes par client, par produit et par mois.

Deuxièmement, au lieu de créer un entrepôt normalisé global à partir duquel est ensuite produits des magasins de données (*data marts*), Kimball préfère la création de magasins de données comme base de l'architecture. Ces magasins, des sous-ensembles de données regroupées généralement pour répondre aux besoins spécifiques d'un département ou secteur de la compagnie, sont assemblés pour créer un entrepôt. Cette approche est possible lorsque les dimensions sont conformes, c'est-à-dire qu'elles représentent exactement la même chose d'un magasin à l'autre.

Troisièmement, un autre élément de la méthodologie de Kimball qui diffère avec celle d'Inmon est qu'elle est menée par les besoins des clients. Le cycle de vie d'un projet utilisant la méthodologie Kimball débute comme celui d'un projet traditionnel en cascade, soit par l'élaboration des besoins du client. Ensuite, Kimball distingue les tâches de création d'entrepôts entre trois spécialisations du monde de l'intelligence d'affaires [22] :

- l'environnement technique,
- les données et
- les applications analytiques.

Les tâches des chemins spécialisés peuvent être exécutées en parallèle, car elles ne visent pas les mêmes éléments de l'environnement.

Le chemin technique est orienté vers le choix et la configuration des outils utilisés pour les entrepôts. C'est aussi dans ce chemin qu'est définie l'architecture technique, donc l'ensemble des éléments matériels et logiciels formant l'environnement d'un entrepôt.

Le chemin des données est emprunté par les programmeurs qui modélisent les magasins de données et acheminent les données des systèmes sources vers ces magasins.

Le chemin pour le développement d'applications analytiques est prévu pour les développeurs de rapports, d'analyses et de tableaux de bord typiquement dans des logiciels prévus pour la consommation de données analytiques.

Ces trois chemins sont intégrés en fin de projet au moment du déploiement. Le processus complet est répété pour chaque nouveau magasin de données demandé par les utilisateurs.

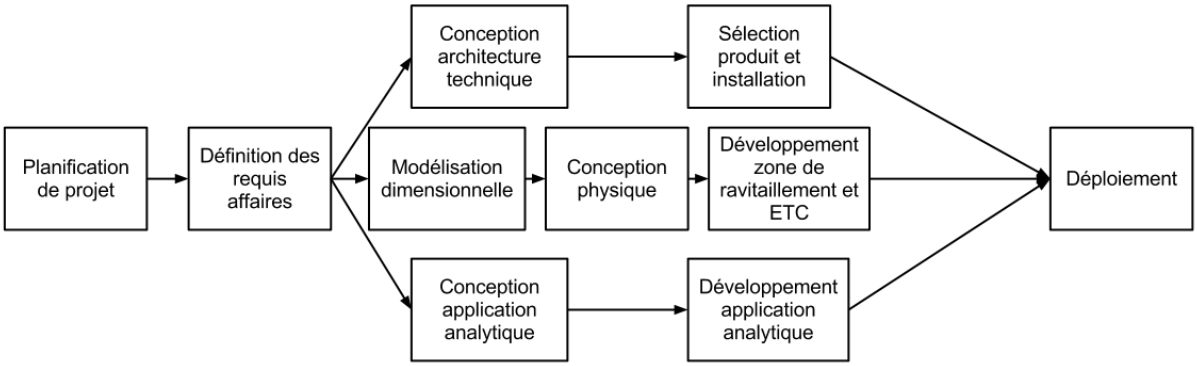

Figure 1.3 Étapes de projet avec la méthodologie Kimball

1.5. Problématique

Malgré l'utilisation de méthodologies formelles, le taux d'échec des projets d'entrepôt demeure élevé. Cette section donne des statistiques et liste les raisons possibles d'échec de ces projets.

Le consortium Cutter [9] a publié des statistiques sur le taux d'échec des projets d'entrepôt. Ce consortium a sondé 142 compagnies et 41 % d'entre elles ont vécu une forme d'échec dans leurs projets d'entrepôt. De plus, seulement 15 % des répondants disent avoir vécu un franc succès avec de tels projets.

En plus des consortiums, certains chercheurs se sont aussi penchés sur les taux d'échec des projets d'entrepôts. Pour sa part, Trembly [42] rapporte que le taux d'échec d'un projet d'entrepôt peut être aussi haut que 90 %.

1.5.1. Les causes d'échec

Dans la littérature scientifique, beaucoup d'auteurs ont cherché à déterminer les causes d'échec des projets d'entrepôt. Les conclusions sont diverses : des auteurs mentionnent les problèmes avec le client et ses besoins imprécis, le manque de qualité des livrables, les problèmes politiques ou budgétaires et enfin, les difficultés de maintenance.

Un projet d'entrepôt, comme la plupart des projets TI, est démarré pour combler un besoin du client. Pour augmenter le taux de succès d'un projet, l'équipe TI doit saisir les besoins du client. Riggle [33] croit que les échecs des projets d'entrepôt sont attribuables à une incompréhension du client par l'équipe TI. En effet, l'équipe TI n'a souvent pas la connaissance tacite de l'entreprise qui est requise pour offrir un produit qui a de la valeur pour le client. Il arrive même que le mauvais système soit créé. [33]

Gharaibeh [13] explique que le fossé de communication entre l'équipe TI et les clients est inévitable, car les individus sont différents à plusieurs points de vue :

- les connaissances,
- les personnalités,
- les habiletés,
- les postes qu'ils occupent et
- les décisions qu'ils doivent prendre.

Trembly [42] stipule qu'une difficulté rencontrée est de traiter avec des utilisateurs de plusieurs départements. Les définitions de termes communs peuvent différer d'un département à l'autre et rendent difficile la consolidation des données.

Zhai [46] décrit les raisons derrière les problèmes de découverte des besoins des utilisateurs. Tout d'abord, les besoins tels que décrits sont déficients, car les utilisateurs sont incapables de

donner leurs exigences complètes. Ensuite, l'utilisateur peut inonder l'équipe TI avec ses besoins en raison d'attentes exagérées quant au projet, ou de la capacité qu'a l'équipe TI de régler l'ensemble de ses problèmes. Enfin, les besoins sont changeants et peuvent entrainer une modification du travail déjà effectué, causant un retard dans le projet. Breur [5] donne une autre raison pour les exigences incomplètes : il semble difficile pour un utilisateur d'imaginer la manipulation d'un système qu'il n'a jamais vu.

Connor [8] écrit que certains clients ne sont pas convaincus ou ne s'impliquent pas suffisamment dans les projets. Cet auteur met en garde les équipes TI contre l'ajout continuel de nouveaux besoins par le client, ces ajouts pouvant entrainer l'échec du projet.

Le manque de qualité des livrables est un thème récurrent dans la littérature. Plusieurs auteurs mentionnent que la qualité des données dans l'entrepôt est un élément clé dont l'absence compromet le projet. [8] [10] [19] [35] [37] [42] La qualité n'est pas seulement un attribut de la donnée, mais en qualifie aussi l'accès. Jukic [19] explique que la documentation des requis est trop technique pour les utilisateurs. Ceux-ci ont tendance à accepter ce qui est documenté sans comprendre exactement ce qui sera fait, dans le seul but de voir le projet se réaliser rapidement. L'effet de cette approbation est la livraison d'un projet avec certaines fonctionnalités imparfaites ou superflues.

La politique interne à l'entreprise peut aussi être un frein au succès d'un projet d'entrepôt. Trembly [42] fait mention de la résistance des directeurs de départements quant à l'accès à leurs données. Ils deviennent très territoriaux, car ils sentent une perte de contrôle de leur environnement décisionnel.

Pour les enjeux de gestion, Trembly fait aussi mention de la difficulté de calculer le retour sur investissement de ces projets. En effet, les projets d'entrepôt ne remplacent normalement pas un processus manuel tel la prise de commande. Ces projets offrent des bénéfices souvent intangibles, rendant ardu le calcul d'un retour monétaire sur l'investissement. De plus, les projets sont souvent trop coûteux et le temps alloué trop court.

Enfin, l'action de régler un problème peut en entraîner l'introduction d'un autre. Bien que ce point problématique se voit surtout en maintenance, il peut également se manifester durant le projet. En effet, Brooks [7] estime entre 20 % et 50 % la probabilité d'introduire un problème quelconque lors d'une correction ou d'un changement.

Parmi ces problématiques, plusieurs peuvent être adressées en modifiant l'approche de l'équipe TI face aux projets. Le reste de cet essai explique une approche pour minimiser ou éliminer plusieurs des problématiques décrites dans ce chapitre.

Chapitre 2
Méthodologies agiles

L'agilité peut se définir comme une souplesse dans l'exécution de mouvements ou une vivacité intellectuelle. Ce qualificatif a été utilisé pour décrire un ensemble de méthodologies créées pour répondre à des problèmes rencontrés à l'utilisation des techniques traditionnelles, tel que l'incapacité de livrer un produit de qualité dans les temps et dans le budget. Ces méthodologies, regroupées sous le nom de méthodologies agiles, sont centrées sur le développement logiciel, mais il sera démontré dans ce chapitre qu'il est possible de les adapter à des projets d'entreposage de données. Ainsi, cet essai propose une façon différente de faire réunissant des pratiques traditionnelles et agiles afin de remédier aux problématiques exposées au chapitre précédent :

- les incompréhensions des besoins du client;
- le manque de qualité des livrables;
- les enjeux de gestion;
- l'introduction de bogues en maintenance.

2.1. Manifeste de développement agile de logiciel

En 2001, des représentants de plusieurs méthodologies de développement logiciel se sont rencontrés pour établir une base commune à leurs méthodologies. Le produit final de cette rencontre fut le manifeste de développement agile de logiciels, [16] qui se lit comme suit :

« Les individus et leurs interactions plus que les processus et les outils.
Des logiciels opérationnels plus qu'une documentation exhaustive.
La collaboration avec les clients plus que la négociation contractuelle.
L'adaptation au changement plus que le suivi d'un plan. » [3]

Tableau 2.1 Caractéristiques des méthodologies agiles

Caractéristiques des méthodologies agiles	
Adaptatif	Une méthodologie qui s'ajuste à la situation. Elle est plus souple dans son exécution.
Itératif	Un processus répété fréquemment pour avoir une rétroaction de l'utilisateur. De plus, le processus est incrémental, car chaque itération est construite sur les précédentes.
Simple	La simplicité d'une méthodologie facilite son adoption et son application sans tracas.
Promotion de la communication	La communication mise en avant-plan dans une méthodologie augmente les probabilités de produire un logiciel conforme aux besoins du client.

Traduction libre

Inspiré de : Maher, P. (2009), p. 1687

Les méthodologies agiles privilégient les individus, les logiciels opérationnels, la collaboration et l'adaptation au changement. À l'opposé, les façons de procéder traditionnelles se concentrent sur les processus et les outils. En considérant ces faits, les représentants ont émis l'hypothèse que les premières peuvent faire mieux que les secondes. De son côté, Kimball recommande de définir les besoins du client avant la modélisation et planifier de façon détaillée pour des projets d'entreposage de données. [22] Cette approche est appropriée

lorsque le client peut définir la majorité de ses besoins clairement, mais elle manque par contre de flexibilité lorsque les besoins sont imprécis. Dans un projet d'entrepôt, il est fréquent que le client ne sache pas exactement ce dont il a besoin, car il n'a souvent qu'une idée imprécise de ce qu'un entrepôt peut lui offrir. [46]

Bien qu'il existe une variété de méthodologies accessibles pour réaliser un projet TI, il n'est pas nécessaire d'évaluer l'ensemble de celles-ci pour avoir un portrait global des tendances en gestion. De plus, l'hypothèse de cet essai est que les méthodologies agiles peuvent réduire les risques d'échec d'un projet d'entrepôt, alors le choix de méthodologies est limité à celles qui sont agiles.

Pour qu'une méthodologie soit considérée dans cet essai, elle doit posséder certaines caractéristiques de base, répertoriées dans le tableau 2.1 et suivre les principes listés dans le tableau 2.2. Parmi les méthodologies qui respectent ces critères, une sélection éclectique a été effectuée pour réduire la portée de l'essai. Les méthodologies étudiées sont :

- *eXtreme Programming;*
- *Scrum;*
- *Lean;*
- *Dynamic System Development Methodology (DSDM);*
- *Agile Modeling;*
- *Feature-Driven Development*.

Une description sommaire de chaque méthodologie est faite dans les sections suivantes.

Tableau 2.2 Liste des principes notables des méthodologies agiles

Principes des méthodologies agiles
Satisfaire le client à travers la livraison en continu de logiciels qui lui apportent de la valeur.
Recevoir les demandes de changement, même si elles arrivent tard dans la conception. Les méthodologies agiles exploitent les changements pour donner l'avantage compétitif au client.
Livrer un logiciel fonctionnel le plus fréquemment possible, idéalement entre quelques semaines et quelques mois.
Les représentants de l'entreprise et les développeurs travaillent ensemble tout au long du projet.
La meilleure façon de transmettre l'information est la conversation face à face.
La simplicité est essentielle; il est plus facile d'en ajouter à un processus trop simple que d'en enlever à un processus trop compliqué.

Traduction libre

Source : Beck, K. (2001), p. 1

2.2. eXtreme Programming

La méthodologie la plus connue du mouvement agile est *eXtreme Programming* (XP). Elle a été développée par Kent Beck dans le cadre d'un projet chez Chrysler. [4] Cette approche contient un ensemble de valeurs, d'activités et de pratiques. Une valeur est un principe par lequel le programmeur doit agir, une activité est une étape concrète dans la méthodologie et une pratique est ce qui est fait durant les activités. Ces éléments sont décrits dans les prochains paragraphes.

XP établit un cadre pour améliorer le développement logiciel en améliorant la qualité et la capacité de répondre aux besoins changeants des utilisateurs. C'est en acceptant que le client puisse changer d'idée dans le projet et en mettant en place les mécanismes pour aider les changements fréquents que XP permet de livrer ce que le client désire.

2.2.1. Valeurs de XP

Les valeurs viennent en premier lieu et constituent la toile de fond de la méthodologie. [4] La première de ces valeurs est la communication. Elle se fait le plus directement possible, en éliminant les intermédiaires. Le but est de réduire la distorsion du message entre le client et le programmeur, pour que le produit final respecte les besoins du premier.

Dans des projets d'entrepôt, l'information constitue le produit final et sa justesse peut être atteinte lorsque le client et l'équipe communiquent adéquatement. Une incompréhension des besoins du client peut influencer le succès du projet. La communication au sein de l'équipe est aussi importante pour que les membres de l'équipe travaillent vers un but commun.

La deuxième valeur est la rétroaction. Il est dit de la rétroaction qu'elle aide à l'apprentissage, surtout si elle est offerte en temps opportun. [31] Une rétroaction concrète de l'état du système est inestimable. [4] Elle permet de s'assurer que ce qui est fait est bien fait. De plus, elle sert de mesure d'achèvement de la tâche.

La troisième valeur est la simplicité. Celle-ci se manifeste dans toutes les pratiques et activités de XP. La méthodologie tient pour acquis qu'il est préférable de concevoir le code de la façon la plus simple possible, au risque de le modifier plus tard si le besoin survient, plutôt que de recourir à un concept complexe qui pourrait ne pas être utilisé. [4] Cependant, simplicité ne signifie pas naïveté. Il existe des patrons de conception complexes en entreposage de données, tels que ceux présentés par Kimball dans *The Data Warehouse Toolkit*, [22] qui résolvent des problèmes précis d'entreposage afin d'éviter de réinventer la solution à chaque projet. L'utilisation d'un patron de conception peut sembler complexe mais s'avère souvent la solution la plus simple pour résoudre un problème précis.

La dernière valeur de XP est le courage. Cette valeur est requise d'abord et avant tout pour adopter les autres valeurs et les pratiques de XP. [4] Le courage est nécessaire pour utiliser une conception simple, au lieu de prévoir en fonction de besoins indéfinis qui pourraient survenir ou pas. D'autre part, il en faut également pour faire face à la critique qui découle de la

rétroaction. De plus, le programmeur fait preuve du courage pour admettre un manque dans sa compréhension du besoin et pour communiquer ce manque au client. Enfin, le courage s'illustre également par la capacité de jeter du code à la fin de la journée et à tout recommencer le lendemain si celui-ci n'est pas à la hauteur.

2.2.2. Activités de XP

En plus de respecter les valeurs préconisées par XP, le programmeur qui a recours à cette méthodologie répète une série d'activités, à savoir : concevoir, programmer, tester et écouter.

La figure 2.1 illustre ce cycle d'activités. Tout d'abord, il faut concevoir. Dans cette activité, le programmeur imagine le prochain élément à créer dans le projet, ce qui peut être la prochaine classe à créer ou le nouveau rapport à générer. Ensuite, le programmeur concrétise ce qu'il vient de concevoir; selon le cas, ce peut être du code pour un développement de logiciel ou la configuration d'un processus d'ETC, par exemple, dans un projet d'entreposage de données. Par la suite, le même programmeur se doit de tester l'ensemble du logiciel déjà codé, à l'aide des tests unitaires automatisés (2.2.8) et de l'intégration continue (2.2.12). En terminant, le programmeur doit écouter le client pour planifier la prochaine itération.

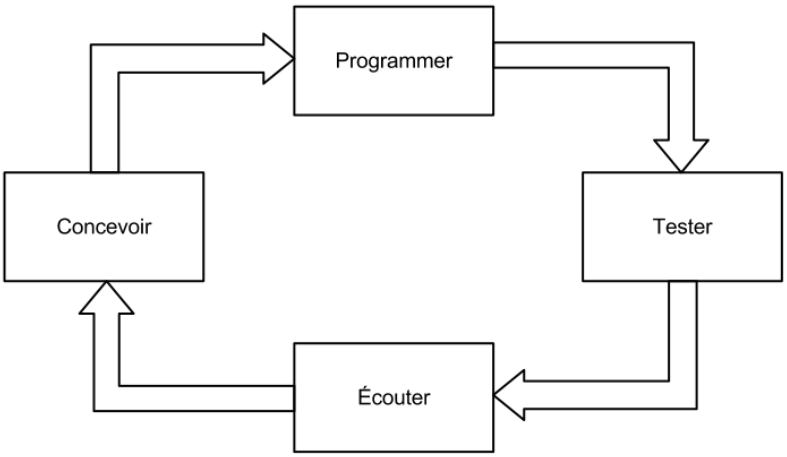

Figure 2.1 Activités de XP

2.2.3. Pratiques de XP

Les pratiques de XP complètent la méthodologie. L'idée directrice de XP est de prendre des pratiques déjà acceptées pour leur efficacité et les appliquer de façon extrême. [4] Par exemple, il est reconnu que la révision de code est une bonne façon d'améliorer le code. [4] La version extrême de la révision est la programmation en paires, où le code est examiné au moment même où il est écrit. La logique est similaire pour toutes les pratiques de XP.

Bien que XP ait été conçu dans le contexte d'un projet de programmation orientée-objet, ses valeurs et pratiques semblent suffisamment universelles pour être adoptées dans un projet d'entrepôt; c'est pourquoi elles seront utilisées dans l'élaboration de la méthodologie décrite dans cet essai. Les pratiques de XP sont détaillées dans les sections suivantes.

2.2.4. Le jeu de la planification

Le jeu de la planification se pratique en début de projet et avant de commencer chaque itération. Pour ce faire, les clients et les membres de l'équipe TI se réunissent pour déterminer le contenu de la prochaine itération. [4] Les clients sont responsables de la portée, de la priorité, du contenu de la prochaine livraison et des dates de livraison. Dans un projet d'entrepôt, les clients peuvent être représentés par des experts techniques, ces « personnes capables de décrire en détail les différentes tâches d'un emploi donné et les connaissances et capacités requises pour exécuter ces tâches. » [27] De leur côté, les membres de l'équipe TI informent les clients des estimations et des conséquences de leurs choix. De plus, les membres de l'équipe TI décident des processus et de l'ordre de réalisation du contenu de la prochaine livraison.

Ensemble, les clients et les membres de l'équipe TI s'entendent sur le déroulement de la prochaine itération. Malgré tout, le plan n'est pas immuable. En effet, si un évènement susceptible de changer ce plan survient, la modification n'est effectuée qu'avec l'accord des deux parties.

2.2.5. Petits livrables

XP recommande des itérations aussi courtes que possible. Avec une durée limitée, le client doit déterminer ce qu'il souhaite obtenir dans un livrable pour ensuite ne sélectionner que ce qui lui semble le plus important.

Le livrable doit cependant contenir des fonctionnalités complètes afin d'offrir une plus grande valeur compte tenu des besoins du client. [4] Par conséquent, dans le jeu de la planification, les parties prenantes s'assurent de l'intégrité du livrable. De plus, les besoins du client peuvent demeurer identiques durant la courte durée de l'itération.

2.2.6. Métaphore

Une métaphore se veut un scénario partagé par l'ensemble de l'équipe. Ce scénario consiste à guider les programmeurs en expliquant le fonctionnement du système. [4] Par exemple, une métaphore pour un site de commerce en ligne serait le panier. Lors d'une séance de magasinage, chaque item désiré est mis dans le panier d'achats. À la fin des achats, le contenu du panier est facturé.

Une métaphore amène la transposition des fonctionnalités abstraites d'un logiciel par des concepts tangibles qui en simplifient la compréhension. De plus, avoir une vision commune du système à concevoir facilite la communication dans l'équipe.

2.2.7. Conception simple

Une conception simple est atteinte lorsque le programmeur ne code que les classes respectant les caractéristiques suivantes : [4]

- exécuter tous les tests correctement;
- éliminer la duplication de logique;
- respecter l'intention du programmeur;

- contenir le plus petit nombre possible de classes et de méthodes.

Le développeur d'entrepôt, par exemple, limite à l'essentiel l'inclusion de mesures dans une table de faits. Tout élément superflu est alors éliminé, rendant le système plus simple et plus facile à tester.

2.2.8. Tester

Dans la méthodologie XP, deux variétés de tests existent : les tests unitaires automatisés et les tests fonctionnels. Les premiers sont écrits et exécutés par les programmeurs alors que les seconds sont générés par le client. [4] Les tests unitaires automatisés sont codés par les programmeurs, avant d'écrire le code de production, afin d'assurer le fonctionnement des classes et méthodes.

Le fonctionnement doit respecter les spécifications du client, de même que la métaphore et les attentes du programmeur. De cette manière, les tests peuvent être exécutés à chaque compilation. Les cas de tests servent à communiquer l'intention du programmeur lors de l'élaboration du code car le programmeur utilise le code de production explicitement avec ses cas de tests. Un test unitaire donne une rétroaction efficace au programmeur en lien avec l'avancement de sa programmation si tous les tests unitaires sont concluants, et si aucun autre test ne doit être exécuté, le programmeur a terminé de rédiger son code. Lorsque tous les tests unitaires sont exécutés avec succès, le système est complet et prêt à être livré.

Cette pratique est à la base d'une autre méthodologie agile, nommée *Test-Driven Development* (TDD). La création d'un processus ETC est souvent limitée à une configuration visuelle qui n'offre pas de langage de programmation formel. Utiliser TDD dans ce contexte est un défi étant donné que le programmeur ne peut souvent pas utiliser les outils de développement pour exécuter les tests. Pour un ETC, un test unitaire pourrait être la génération d'un jeu de données standard qui serait utilisé à chaque modification du processus dans le but de comparer les résultats de l'ETC avec ceux escomptés par le test. En somme, tester tôt et tester souvent sont les pratiques XP pour tout projet. [4]

2.2.9. Réusinage

Le réusinage se définit comme un exercice de restructuration du logiciel dont l'objectif est de simplifier, d'éliminer la duplication de code ou de fonctionnalités, d'améliorer la communication ou d'ajouter de la flexibilité. [4]

L'ajout de flexibilité est réalisé lorsque le besoin se présente car le programmeur n'élabore au départ que le code jugé nécessaire. Néanmoins, le système doit conserver le fonctionnement prévu tel que décrit par les tests car ceux-ci constituent un filet de sécurité pour le programmeur qui performe le réusinage. Le système est réusiné avec succès que lorsque l'exécution des tests est concluante.

2.2.10. Programmation en paires

La programmation en paires, deux programmeurs qui codent sur le même poste de travail simultanément, est la version extrême de la révision de code et se caractérise par une vérification simultanée du code lors de l'écriture du second programmeur. [4] Le résultat de cette pratique est une rétroaction instantanée sur la qualité du code et le respect des standards. De plus, la présence de deux programmeurs à un même poste peut contribuer à la conception simple lorsque ceux-ci ont une opinion similaire sur la manière la plus simple de procéder. Généralement, un programmeur, le conducteur, a le contrôle du poste de travail. Il programme pendant que son acolyte, l'observateur, a la responsabilité de les guider vers leur but commun : répondre aux besoins du client.

La programmation en paires dans un projet d'entreposage de données peut être exemplifiée par la mise au point d'un rapport par le conducteur, pendant que l'observateur vérifie le respect des règles et du format préétabli de présentation visuelle.

2.2.11. La communauté est propriétaire du code

Dans un système testé continuellement et où le réusinage du code est encouragé, le code doit être accessible par tous les membres de l'équipe TI. [4] Lorsque le code était la propriété d'un

programmeur en particulier, les autres programmeurs doivent faire des demandes de changement à ce programmeur. Le code reste stable, mais n'évolue pas aussi rapidement car les autres programmeurs sont réticents à déranger le programmeur en charge. Par conséquent, il y aurait une perte de flexibilité et une diminution de la connaissance de l'équipe du code ce qui pourrait même empêcher l'atteinte du but des autres pratiques.

Par exemple, le réusinage peut difficilement être accompli sans avoir la liberté de modifier le code. Éliminer l'appropriation du code par des membres spécifiques de l'équipe rend le code communautaire, donc tous les membres sont propriétaires du code. En procédant ainsi, le code devient la propriété de tous les membres de l'équipe, ce qui contribue à l'autonomisation de chaque programmeur qui agit à chaque occasion de l'améliorer. Cependant, cette pratique ne doit pas être appliquée seule, car le chaos pourrait s'installer dans la base de code. En effet, il est recommandé par Beck [4] d'avoir en place les tests unitaires automatisés, les standards et l'intégration continue comme assurance contre l'introduction de défauts par n'importe quel programmeur.

Appliquer la pratique du code communautaire est possible pour un projet d'entrepôt. Du code peut être présent, mais le paysage d'un tel projet est plus vaste : des ETC, des bases de données, des rapports et analyses, des cubes OLAP et des tableaux de bord. Par conséquent, inclure tous ces éléments dans le sens large de « code » rend cette pratique applicable.

2.2.12. Intégration continue

Dans un projet traditionnel, chaque programmeur œuvre sur une partie du système, indépendamment des autres programmeurs. L'intégration, qui constitue l'assemblage des parties et l'exécution des tests, est exécutée en fin de projet. L'assemblage peut cependant révéler des problèmes : la quantité de code à corriger peut être énorme et nécessiter la réécriture d'une grande partie du système. Pour éviter cette situation, il est suggéré dans la méthodologie XP d'intégrer continuellement à la base commune le code que le programmeur vient d'écrire et de tester. Ceci a pour effet de détecter les problèmes par le programmeur qui

les a causés et de les régler immédiatement. [4] Le résultat de cette démarche est un logiciel fonctionnel et prêt à être déployé en tout temps.

Un entrepôt peut ressembler à une chaîne de production, commençant au système opérationnel et terminant au rapport ou au tableau de bord. Chaque maillon de la chaîne est une action exécutée sur une donnée, et influence le résultat final. Lorsque l'un de ces maillons est défectueux, le résultat final est erroné. Intégrer continuellement les maillons de cette chaîne permet de déceler promptement une faille dans le processus. Comme pour le développement logiciel, l'intégration continue doit se faire avec les tests unitaires automatisés, sans quoi il est difficile de garantir une intégration réussie.

2.2.13. Semaine de quarante heures

La règle de la semaine de quarante heures est d'éviter de faire des heures supplémentaires et, surtout, de ne pas en faire deux semaines consécutives. Bien que Beck recommande quarante heures, il mentionne que chaque individu a sa propre tolérance pour la durée de sa semaine de travail. Certains ne sont productifs que trente-cinq heures alors que d'autres peuvent en travailler quarante-cinq. [4]

Il se peut que des heures supplémentaires soient requises dans un projet une semaine ou deux, mais un retour aux heures normales doit se faire par la suite; personne ne peut être plein d'entrain et créatif lorsqu'il travaille soixante heures par semaine, toutes les semaines. [4]

2.2.14. Client sur place

La programmation se caractérise par l'élaboration d'une solution à un besoin de l'utilisateur. À chaque fois qu'un programmeur analyse le besoin à combler, il en précise sa compréhension et découvre un déficit dans sa connaissance. C'est alors qu'il communique avec le client pour combler cette lacune informationnelle.

Lorsque le client n'est pas disponible pour répondre aux interrogations du programmeur, deux choix s'offrent à ce dernier : attendre que le client soit disponible, ou supposer de sa réponse et

continuer à programmer. Dans le premier cas, le projet est retardé par une diminution de la performance du programmeur alors que dans le second cas, le programmeur peut se tromper et introduire une faille dans le système. Pour éviter ces problèmes, il est recommandé que le client soit sur place avec les membres de l'équipe TI [4] afin qu'il soit disponible en tout temps pour aligner l'évolution du système avec les besoins du client et diriger l'équipe vers un logiciel répondant à ses attentes.

Comme mentionné plus haut, le client peut être représenté par un expert technique qui serait sur place, au sein de l'équipe TI, pour offrir ses connaissances dans le domaine d'affaires et donner une rétroaction nécessaire à l'orientation efficace des efforts de l'équipe TI.

2.2.15. Standard de programmation

Se doter d'un standard de programmation et le mettre en pratique par l'équipe est essentiel pour saisir l'intention du programmeur à la seule lecture de son code. [4] De plus, dans un projet où tous les membres sont encouragés à améliorer constamment le code, l'application uniforme d'un standard est impérative. Le but est d'éviter qu'un programmeur utilise son propre style d'écriture. Les standards doivent également être respectés dans les autres sphères du projet : l'outil d'ETC à utiliser, le style de présentation de rapports et la nomenclature des bases de données sont des exemples de standards à respecter dans l'univers de l'entreposage de données.

Les valeurs, les activités et les pratiques de XP ont été développées pour répondre aux besoins des équipes de développement de logiciel. Malgré tout, la philosophie de XP est applicable dans d'autres projets TI, car ses valeurs ne concernent pas que le développement de logiciel, et sont applicables dans d'autres secteurs des TI. Ainsi, ces valeurs peuvent être mises en œuvre dans un projet d'entrepôt. Les activités de XP sont comparables à celles d'un projet d'entrepôt si le sens de « programmer » est élargi pour inclure les opérations propres à un projet d'entrepôt.

2.3. Scrum

La méthodologie Scrum a été développée par Ken Schwaber, en collaboration avec Jeff Sutherland. Elle repose sur un cycle de livraison de trente jours surnommé le sprint. [36]

Schwaber et Sutherland soutiennent qu'on ne peut connaitre la valeur d'un logiciel pour les besoins du client tant que ce logiciel n'est pas testé, intégré et livré. La rétroaction du client est possible tous les trente jours, une fois la livraison effectuée. Au début de chaque sprint, l'équipe TI et le client choisissent des fonctionnalités à concevoir à partir du carnet du produit. Une réunion quotidienne de 15 minutes, appelée mêlée, rassemble l'équipe pour constater l'avancement du projet et s'adapter aux problèmes vécus. À la fin du cycle, l'équipe présente les fonctionnalités complétées au client. En définitive, Scrum est une méthodologie centrée sur la gestion et peut être applicable à des projets TI pouvant être livrés de manière incrémentale.

2.4. Lean

Mary et Tom Poppendieck ont développé la méthodologie *Lean* en utilisant le système de développement de produits de Toyota comme fondation. [30] La méthodologie est empirique, ce qui lui permet de s'adapter aux changements durant le projet. La méthodologie déconseille une planification exhaustive pour des éventualités peu probables afin de réduire le gaspillage de ressources monétaires et humaines. [30] Les sept principes qui constituent *Lean* sont décrits dans les points suivants.

2.4.1. Éliminer le gaspillage

Un des principes les plus importants dans *Lean* est l'élimination du gaspillage : enlever tout élément qui ne permet pas au produit de combler un besoin du client. [30] Une liste des formes de gaspillage selon *Lean* se retrouve en annexe 1.

Ces formes de gaspillage se retrouvent à toutes les étapes d'un projet et requièrent une analyse détaillée du processus de projet pour les faire disparaitre. Par exemple, dans un projet d'entrepôt, inclure des mesures dans une table de fait qui ne sont pas requises par le client,

« juste au cas », est une forme de gaspillage car il est possible que le client ne les utilise jamais. Il s'agit donc de l'utilisation inutile (au sens de *Lean*) de ressources.

2.4.2. Développer la qualité à l'interne

Développer la qualité à l'interne vise à garantir la qualité du code dès le départ et non seulement le tester par la suite. Effectivement, il est moins coûteux d'éviter de créer des défauts que de les détecter ultérieurement. [30] Étant donné que programmer peut introduire des défauts, il est préférable d'inspecter le code en appliquant des pratiques empruntées de XP, soit l'intégration continue et les tests automatisés.

Dans une chaîne de production industrielle, la qualité du produit fini dépend de celle de la matière première. Pour un projet d'entreposage de données, la matière première est la donnée venant des systèmes sources. Développer la qualité à l'interne, dans ce contexte, signifie intégrer des tâches de validation des données aux étapes de transformations.

2.4.3. Créer le savoir

Le développement logiciel représente un processus de création du savoir [30] acquis par rétroaction. Dans ce cas, le programmeur sait qu'il a bien effectué son travail lorsque le client émet ses commentaires.

La conception initiale au moment de l'élaboration d'un projet n'est pas propice à la rétroaction du client. Certes, la forme abstraite de la conception freine la compréhension par la clientèle. Par contre, la rétroaction se fait aisément quand le client peut interagir avec le logiciel fonctionnel. En définitive, pour créer le savoir, il faut des boucles fréquentes de rétroaction.

2.4.4. Différer l'engagement

Chaque décision prise dans un projet comporte un coût, et chaque décision renversée entraîne un coût encore plus important. Par conséquent, retarder la prise de décision est susceptible

d'entraîner une économie. [30] Dans l'impossibilité de différer l'engagement, il est souhaitable de rendre un maximum de décisions réversibles afin de minimiser les coûts.

Par contre, une balance doit être établie entre prendre une décision hâtive et la prendre trop tard. Une décision hâtive peut empêcher un jugement plus éclairé lorsque tous les éléments requis pour prendre le bon choix sont connus. Malgré tout, il se peut que l'atteinte du moment propice ne soit pas possible et que l'absence de décision mette en péril un projet.

2.4.5. Livraison rapide

Un des buts de *Lean* est de « livrer assez rapidement le logiciel pour que le client n'ait même pas l'opportunité de changer d'idée ». [30]

Cependant, une livraison prompte doit s'exécuter avec un respect pour la qualité du logiciel. L'idée derrière la livraison rapide n'est pas de donner au client le premier jet d'un programme en espérant qu'il réponde à ses besoins; il s'agit plutôt d'établir un processus suffisamment léger pour que le temps séparant la demande de la livraison soit si petit que le client puisse conserver ses besoins initiaux.

2.4.6. Respecter l'individu

L'individu doit être au centre du processus créatif. Idéalement, l'équipe doit avoir un chef entrepreneurial et une main d'œuvre technique experte. La direction attribue à l'équipe un plan général et des buts atteignables. Par conséquent, la direction respecte l'équipe et ses choix en lui permettant d'être autonome dans son organisation. [30]

2.4.7. Optimiser la chaîne

Le processus complet du développement logiciel est une chaîne de production débutant par la demande d'un client et se terminant par la livraison. Lorsque la chaîne n'est pas optimisée, du gaspillage et des délais surviennent. Il faut tout mettre en œuvre pour maximiser l'efficacité la chaîne de production afin de livrer le logiciel plus rapidement. [30]

En somme, *Lean* adapte les concepts de chaîne de production et de gaspillage au domaine des TI. Un projet d'entreposage de données peut aisément être perçu comme une chaîne de production, avec les données de systèmes opérationnels comme matière première et le cube OLAP comme produit fini.

2.5. Dynamic System Development Methodology

Un consortium de compagnies international a créé la méthodologie *Dynamic System Development Methodology* (DSDM) au milieu des années 90. [38] Pour ce faire, le consortium a combiné les techniques de *Rapid Application Development* (RAD) avec les principes de Scrum. [39] Avec RAD, le programmeur crée des prototypes d'interfaces utilisateurs pour avoir une rétroaction du client, et ainsi combler ses besoins.

La figure 2.2 illustre les étapes d'un projet DSDM. Premièrement, l'étude de faisabilité lors de laquelle il est déterminé si DSDM peut être utilisé dans le projet. Deuxièmement, le dossier décisionnel cible les besoins de haut niveau. Troisièmement, le modèle fonctionnel itératif est implémenté et contient la définition architecturale, les spécifications de produits et les prototypes. Quatrièmement, le logiciel est créé dans la phase de conception. Cinquièmement, le logiciel est mis en œuvre. Excluant l'étude de faisabilité, les phases de DSDM sont itératives et le projet peut passer d'une étape à l'autre à maintes reprises. [43]

Pour conclure, DSDM améliore le RAD en employant des principes de Scrum. Cette méthodologie est spécifique aux applications ayant une interface graphique limitant l'applicabilité à une partie des projets d'entrepôt : la portion d'accès des données par l'utilisateur.

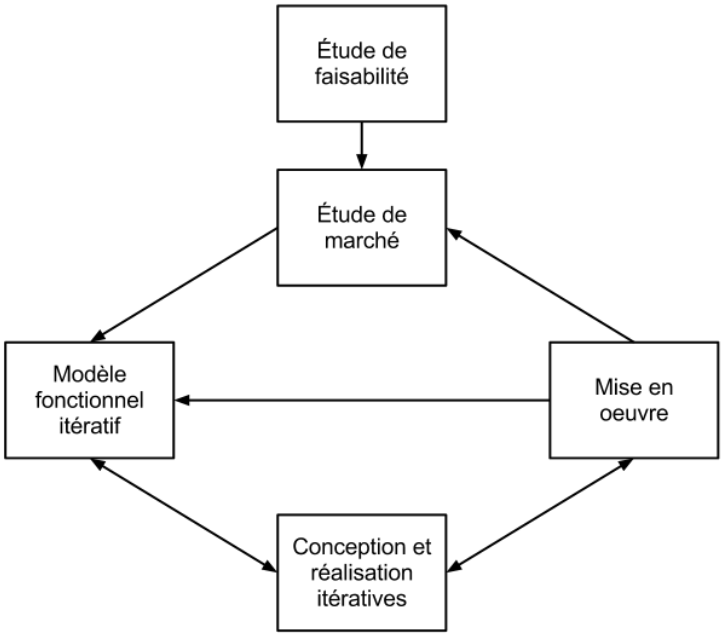

Figure 2.2 Étapes d'un projet avec DSDM

Traduction libre

Source : Wikipédia. (2012)

2.6. Agile Modeling

La méthodologie *Agile Modeling* (AM) a été développée pour être intégrée à d'autres méthodologies. [2] AM a comme principe de présumer la simplicité et d'embrasser le changement. Il est plus facile de refléter un changement dans un modèle si celui-ci est simple. Il est prescrit dans cette méthodologie de modéliser avec un but : connaître le public visé et les besoins de ce dernier. Si le développeur ne sait pas pourquoi ou pour qui le document doit être créé, il doit s'abstenir de le produire. AM est une philosophie sur la création de modèles, au même titre que XP l'est pour la programmation. Il n'y a pas de styles de modèle préétablis avec cette méthodologie, pas plus qu'il n'y a de cadre formel; rien d'autre qu'une approche légère sur la documentation qui doit être produite. Par exemple, dans le but de cerner la portée

et l'architecture probable du système, il est préférable d'effectuer une période d'analyse de haut niveau au début du projet. Au démarrage d'une itération, une étape de modélisation accompagne la planification du livrable. AM suggère la modélisation *Just-in-Time* pour éviter de produire un document qui ne sera pas utilisé. La modélisation peut se dérouler durant la construction du livrable et avant les activités de codage, c'est-à-dire au moment où le programmeur requiert le modèle.

En somme, AM vient compléter les autres méthodologies en rendant la modélisation explicite. Pour des projets d'entrepôt la modélisation est primordiale.

2.7. Feature-Driven Development

Feature-Driven Development (FDD) emprunte plusieurs pratiques des autres méthodologies. À la base, cette méthodologie a été créée pour des programmeurs Java, mais a depuis été modifiée pour être bénéfique à d'autres domaines des TI. Un projet se déroulant avec FDD débute avec la création d'un modèle d'objets de domaine cartographiant l'ensemble du système à développer. Ensuite, les programmeurs font une liste des fonctionnalités à implémenter et un plan rudimentaire est élaboré pour amorcer les itérations. Enfin, la conception se fait une fonctionnalité à la fois.

La fonctionnalité est au centre de cette méthodologie. Pour faire un parallèle avec un projet d'entrepôt, la fonctionnalité pourrait être une dimension, ou encore une table de faits, incluant les processus d'ETC qui les alimentent.

Chapitre 3
Méthodologie proposée

Le premier chapitre de cet essai introduit les méthodologies traditionnelles de conception d'entrepôt de deux auteurs reconnus dans le monde de l'intelligence d'affaires. Brièvement, Inmon préconise d'analyser l'entrepôt pour tous les besoins de l'entreprise au départ, pour s'assurer que cet entrepôt contienne tous les éléments standardisés. À l'opposé, Kimball suggère de créer des magasins de données qui formeront l'entrepôt au cours de projets successifs. La méthodologie Kimball est itérative et incrémentale, mais reste traditionnelle dans son exécution avec des tâches en cascade.

Ces méthodologies peuvent être utilisées lorsque le besoin du client est clair. Par contre, une méthodologie agile est requise pour s'adapter aux besoins imprécis et changeants du client. L'auteur du présent essai propose ainsi une approche combinant certains éléments de Kimball avec des pratiques empruntées des méthodologies agiles. Pour ce faire, la méthodologie agile en intelligence d'affaires (MAIA) est décrite dans les prochaines sections.

En premier lieu, MAIA ne tente pas de remplacer tous les aspects d'un projet d'entrepôt. Par exemple, le chemin technique de Kimball, qui se concentre sur l'aspect des serveurs et autres considérations technologiques, est un travail complexe. En effet, les tâches pour déterminer les besoins de l'entreprise en équipement des projets d'entrepôt peuvent durer plusieurs mois. Également, un projet peut être entièrement dédié au choix des outils pour les ETC et de la couche de présentation des données. Aucun bénéfice ne peut être attendu si ce projet est agile car les tâches qui font normalement parties de ce genre de projet sont souvent dans un ordre précis et exécutées en cascade. Par contre, il est essentiel d'avoir l'architecture technologique en place pour que le chemin de modélisation dimensionnelle et le chemin de conception

d’applications analytiques puissent profiter de flexibilité dans leur exécution, sans être contraint à des limitations dues à des enjeux techniques.

De plus, MAIA peut être employée dans des projets de création de nouveaux magasins de données ou dans la maintenance des magasins existants. Dans la littérature technique, les méthodologies décrivent le projet idéal comme étant un logiciel commencé à partir de zéro et se complétant après plusieurs itérations. Cependant, dans un projet d’entrepôt, il est plus commun de bâtir un magasin de données avec des dimensions déjà existantes. On retrouve souvent des projets ne servant qu’à améliorer un magasin existant, soit pour y ajouter des dimensions ou de nouvelles mesures. Ainsi, pour être vraiment utile, une méthodologie doit s’adapter autant à la création de nouveaux magasins de données qu’à la maintenance de ces derniers.

Enfin, les membres de l’équipe qui utiliseront MAIA doivent avoir une ouverture d’esprit afin de vouloir faire les choses différemment. En effet, cette originalité leur permettra d’innover en adoptant des techniques telles que de réaliser des tâches en paire, d’avoir le client sur place ou d’inclure le programmeur dans la phase de planification.

Par conséquent, MAIA emprunte des pratiques des méthodologies agiles de XP, Scrum et *Lean*. Les prochaines sections décrivent les pratiques appliquées dans MAIA et la manière de les adapter à un projet d’entrepôt. Ensuite, une justification en provenance de la littérature scientifique à l’application de ces pratiques valide la méthodologie. Enfin, le cycle de vie d’un projet avec MAIA est illustré.

3.1. Le jeu de la planification

Le jeu de la planification pour MAIA est effectué en deux temps. Premièrement, une réunion préparatoire avec l’équipe TI est demandée pour faire une première évaluation des besoins du client. Cette réunion est nécessaire afin de comprendre les questions que l’entrepôt devra répondre et les besoins analytiques qui devront être comblés par l’entrepôt, car un projet d’entrepôt peut sembler simple pour un utilisateur, mais parfois un besoin utilisateur mineur

peut représenter un changement majeur dans l'entrepôt. De plus, l'équipe doit déterminer quelles seront les sources de données requises. Le résultat de la réunion est la liste de demandes avec une estimation de l'effort nécessaire pour répondre à chaque demande du client.

La seconde rencontre réunit l'équipe TI et le client. Durant cette réunion, l'équipe TI présente l'effort requis par les demandes du client, alors que celui-ci exprime ses priorités. Il s'ensuit une négociation pour déterminer ce que contiendra la prochaine itération. L'équipe TI annonce sa capacité totale pour la prochaine itération, et le client sélectionne les demandes en fonction de l'effort estimé par les TI.

La création d'un magasin de données peut prendre plus d'une itération. Dans ce cas, certaines tâches sont faites dans une itération et les autres sont reportées dans une itération suivante. Une itération ne signifie pas un déploiement : il peut y avoir plusieurs itérations avant de déployer une version d'un entrepôt ou d'un magasin au client. Plusieurs situations peuvent amener à compléter une itération sans un déploiement chez le client. Il y a des itérations techniques qui mettent en place une nouvelle technologie, un mécanisme particulier ou même une plateforme complète et ne sera pas installée en production sans une itération avec un livrable répondant à un besoin client. Certains clients peuvent aussi choisir de ne pas installer chaque itération due à des contraintes particulières. Dans une entreprise qui possède une équipe d'assurance de la qualité, l'équipe TI peut rendre disponible le contenu de chaque itération pour des tests d'acceptation sans pour autant livrer au client.

3.2. Petits livrables

Un entrepôt contient un ensemble de composantes : une base de données, des ETC, des cubes OLAP, des rapports, des tableaux de bord et autres. Chacune de ces composantes peut se retrouver dans un livrable. De plus, on peut y retrouver des procédures d'installation et de chargement initiaux, des configurations de logiciels et toute autre métadonnée nécessaire au bon fonctionnement de l'entrepôt.

Certaines parties peuvent être livrées indépendamment des autres, par exemple un rapport sur un entrepôt existant. Cependant, une majorité des éléments sont interdépendants. Un ETC ne peut s'exécuter sans sa base de données de destination, tout comme un rapport ne peut s'exécuter sans son entrepôt. En ce sens, la décision de définir le contenu d'un livrable doit être prise conjointement entre le client et l'équipe TI. Le client détermine ses priorités et l'équipe TI cerne le livrable en respectant les dépendances de ses composantes.

Il y a plusieurs avantages à livrer de petites itérations. Pierce [29] prétend que des itérations de trois semaines peuvent réduire drastiquement les risques que le projet déraille. Le client a ainsi un produit fonctionnel rapidement. Grâce à son utilisation immédiate, le client peut analyser les besoins comblés par le livrable et déterminer ce qu'ils représentent concrètement, c'est-à-dire qu'il apprend la manière selon laquelle ses besoins exprimés sont implémentés. Plutôt que réaliser les demandes du client de manière chronologique selon sa liste initiale, ses besoins sont priorisés pour lui apporter une plus grande valeur. De plus, l'utilisation du logiciel permet de mettre en lumière les bogues ou malentendus lors de l'élaboration des requis. [12] D'autre part, il est difficile pour le client de vraiment comprendre ce qu'il veut du produit au début d'un projet. [12] En somme, la rétroaction du client lors de la livraison de petites itérations réduit les risques du projet.

3.3. Tester

Cet essai distingue trois types de tests. Premièrement, les tests unitaires servent à vérifier chaque composante de manière isolée. Deuxièmement, les tests d'intégration sont utilisés pour s'assurer du bon fonctionnement du système au complet. Troisièmement, les tests de régression sont exécutés après un changement pour déterminer que le système fonctionne toujours correctement. [15]

Ces tests peuvent aussi être catégorisés en fonction de la portion ciblée : les tests d'arrière-scène visent les ETC alors que les tests d'avant scène visent les rapports, les tableaux de bord et les cubes OLAP.

Plusieurs aspects doivent être testés durant une itération. Tout d'abord, il faut tester les fonctionnalités et confirmer qu'elles répondent aux besoins du client. Ensuite, les utilisateurs interagissent avec le produit pour déterminer sa convivialité. De plus, certains tests doivent mesurer la rapidité d'exécution du produit, que ce soit dans des conditions d'utilisation normale ou durant les périodes de charge élevée des serveurs. Ce test peut être le temps que prend un rapport à être généré ou la durée d'exécution d'un ETC. Enfin, la sécurité doit être testée pour s'assurer que les utilisateurs qualifiés peuvent accéder à l'information tout en empêchant les utilisateurs exclus d'y accéder.

Trois catégories de testeurs sont reconnues dans MAIA : les programmeurs, les responsables d'assurance de la qualité et les utilisateurs. Les programmeurs ont la responsabilité de générer les tests unitaires. En effet, ils sont tenus de tester la fonctionnalité et la performance de chaque composante. Ces tests pourront être inclus dans la batterie de tests de régression. Les utilisateurs peuvent aussi tester les fonctionnalités mais ont également à tester la facilité d'utiliser le produit et de pouvoir répondre à leurs questions. Les responsables d'assurance de la qualité doivent tester l'ensemble de la solution. Impliquer les utilisateurs dans les tests peut être considéré comme une forme de spécification des besoins, surtout si les tests sont développés au début de l'itération. [1] En effet, l'action de créer des scénarios de tests par les utilisateurs oblige un processus cognitif qui se traduit par une précision des spécifications.

L'aspect le plus important des tests dans MAIA est l'intégration de l'assurance qualité au développement. Tester durant le développement évite de cumuler les défauts à la fin de l'itération ou du projet en les détectant rapidement. Cependant, tester est une tâche qui peut prendre plusieurs heures de travail pour le programmeur dans une semaine. Pour alléger le travail du programmeur, MAIA recommande l'utilisation de tests automatisés. Les tests automatisés offrent deux avantages sur les tests manuels. Premièrement, les tests automatisés peuvent être exécutés en tout temps, ce qui évite que les programmeurs repoussent les tests à la fin du projet. Deuxièmement, les tests automatisés sont reproductibles et ainsi plus aptes à être utilisés dans les tests de régression que les tests manuels.

Avec des tests automatisés, il est possible d'établir que les données d'un entrepôt sont valides. [35] De plus, si les tests sont faits de manière à ce qu'ils puissent être exécutés régulièrement dans l'environnement de production, ils peuvent servir de mesure de qualité de l'entrepôt. Schutte affirme qu'un projet utilisant des tests automatisés avec TDD (Test-Driven Development) peut réduire de 40 à 90 % ses défectuosités qu'avec une technique sans TDD. [35]

Dans un contexte d'entrepôt, il n'est pas toujours possible d'automatiser les tests, même si les logiciels d'automatisation modernes peuvent assister aux tests de rapports et d'outils OLAP. En l'absence de tests automatisés, une stratégie de tests manuels doit être mise en place. Cette approche repose sur la discipline du développeur à exécuter les tests manuels à chaque modification, ce qui peut être un effort considérable sans assistance logicielle. Il y a ici une délicate balance entre l'effort d'automatisation et l'effort de tester manuellement, balance déterminée selon l'expérience et les préférences des membres de l'équipe.

Étant donné la nature imprécise et interdépendante de l'information, il est recommandé d'avoir une suite de tests de régression, car il est possible d'impacter les fonctionnalités en place sans s'en apercevoir avant la mise en opération. [24]

En conclusion, les tests sont une partie intégrale de MAIA. La qualité des données de l'entrepôt, des rapports, tableaux de bord et cubes OLAP doivent être une préoccupation constante.

3.4. Réusinage

Avec les tests unitaires automatisés en place, il est possible de valider que le produit à livrer soit fonctionnel. Ces tests peuvent être exécutés en tout temps. S'assurer que le logiciel est opérationnel est une condition essentielle à un réusinage réussi. [4] Dans les aspects où les tests ne sont pas automatisés, le réusinage doit être appuyé de l'ensemble des tests de régression.

Dans un projet d'entrepôt, le réusinage peut être utilisé pour les raisons suivantes :

- améliorer la performance d'un ETC, un rapport ou un tableau de bord;
- permettre la réutilisation de composantes;
- renforcer la qualité ou la robustesse du produit;
- simplifier les ETC dans le but d'en faciliter la maintenance.

Les bénéfices obtenus avec le réusinage se voient surtout avec les ETC. Réutiliser les composantes dans des projets subséquents réduit le temps requis par le programmeur pour élaborer un ETC tout en augmentant la cohésion. Si les outils de présentation le permettent, la transformation des rapports en gabarits accélère la création de rapports subséquents tout en permettant de conserver un standard.

3.5. Programmation en paires

Lors de l'essai de méthodologies agiles dans des projets d'entrepôt, Goede [14] et son équipe ont utilisé avec succès la programmation en paires. Une de leurs conclusions est que la programmation en paires aide à résoudre les anomalies plus rapidement. Par ailleurs, la programmation en paires peut être utilisée durant les différentes tâches des développeurs. Ainsi, une personne peut concevoir les entrepôts, implémenter des ETC et générer des rapports pendant que l'autre valide les besoins et standards. La présence de deux analystes lors d'entrevues avec le client permet d'extraire plus d'information, car chacun peut comprendre certaines subtilités que l'autre ne saisirait pas.

McMahon [26] argumente que la programmation en paires réduit le coût pour atteindre la qualité pour une ligne de code. Il mentionne qu'avec le coût élevé qu'entraîne la présence de deux programmeurs sur une même tâche, le gain peut sembler contre-intuitif. Cependant, avoir une paire rend le code honnête, ce qui veut dire que la qualité est améliorée. Une paire de programmeurs maintient aussi une vitesse de croisière plus grande qu'un programmeur seul,

ce qui justifie la surcharge d'avoir deux personnes sur la même tâche. Enfin, deux programmeurs peuvent résoudre un problème plus rapidement qu'un seul : le consensus des vues différentes des programmeurs réduit la myopie de conception, évitant une refonte plus tard dans le projet.

De son côté, Williams [45] a fait une analyse économique de la programmation en paires. En comparant la programmation solo et celle en paires, elle arrive à la conclusion que le coût de la programmation en paires est 15 % plus grand en temps que sa contrepartie solo. Cependant, la qualité du livrable est plus grande et les occurrences de refonte après livraison sont nettement diminuées. Donc, elle démontre qu'il y a un avantage économique à utiliser la programmation en paires, car cette pratique ajoute de la valeur en réduisant le bénéfice marginal minimal requis pour la rentabilité du projet.

Dans la méthodologie *Lean*, une des formes de gaspillage à éviter est celui survenant lors du transfert de connaissances lors d'échange de dossiers. La programmation en paires peut contribuer à réduire cette forme de gaspillage en affectant deux personnes sur chaque tâche. En effet, il y a toujours deux personnes qui ont la connaissance, ce qui réduit le risque que le dossier soit transféré à quelqu'un qui n'a pas le savoir requis.

3.6. Intégration continue

L'environnement d'un entrepôt est composé de plusieurs parties distinctes :

- les différents systèmes sources comme les progiciels de gestion intégrés, les systèmes patrimoniaux et le Web,
- les ETC et leurs zones de ravitaillement et
- les cubes OLAP, les rapports et les tableaux de bord.

Cet environnement est en constante évolution grâce aux projets menés par différentes équipes TI dans l'entreprise. Un changement dans un système source peut avoir un impact sur la

qualité de la donnée dans un rapport en aval. L'intégration continue dans ce contexte ne doit pas se limiter à ce que gère l'équipe TI en charge du projet d'entrepôt, mais doit s'appliquer à l'ensemble des projets de l'entreprise. Malheureusement, cette approche idéaliste est difficilement réalisable car il est complexe de coordonner plusieurs équipes TI pour intégrer l'ensemble des projets.

Par contre, l'intégration continue des parties sous le contrôle de l'équipe d'entrepôt est faisable et peut donner les bénéfices de détection hâtive des problèmes. À l'opposé, une intégration unique en fin de projet est souvent ardue et risque d'exposer un ensemble de problèmes difficiles à localiser avec précision. Il peut même être impossible de déterminer les responsables des problèmes, obligeant la personne en charge de l'intégration de désigner un programmeur, possiblement non responsable des problèmes, pour la résolution de ceux-ci.

Un autre avantage de l'intégration continue est la motivation du programmeur, qui voit sa contribution dans un système fonctionnel. [17]

Voici les défis de l'intégration continue dans un projet d'intégration :

- certains logiciels d'ETC n'ont pas de méthode simple pour transférer les configurations d'un serveur à l'autre, rendant peu commode l'intégration continue,
- la modification du schéma de base de données peut être restreinte et empêcher le programmeur de rapidement tester ses changements dans un environnement intégré et
- l'environnement est trop complexe et coûteux pour avoir une version dédiée à l'intégration continue.

En conclusion, il est recommandé de faire l'intégration continue en symbiose avec les tests automatisés dans un environnement se rapprochant le plus du prochain environnement de production pour avoir une rétroaction rapide de l'état de santé du travail accompli et éviter les problèmes rencontrés avec une intégration unique en fin de projet.

3.7. Semaine de quarante heures

Dans une étude [25] visant à évaluer les impacts à long terme de l'utilisation de *Scrum* dans une compagnie de développement de logiciels, Mann et Maurer ont découvert qu'il y a eu une diminution marquée d'heures supplémentaires faites par les employés. L'étude s'est déroulée sur deux ans, pendant lesquels *Scrum* a été introduit avec les pratiques d'intégration continue, de programmation en paires et de tests unitaires.

La raison principale de la diminution des heures supplémentaires des employés fut une baisse de la pression exercée par la direction de la compagnie en réponse à l'implantation de *Scrum*. La direction a réalisé les avantages d'un effort soutenu à long terme sur la qualité du travail des ressources tout en maintenant une quantité perçue de travail similaire.

Il est possible dans un projet que des situations émergent qui demandent l'attention soutenue des programmeurs et requièrent des heures supplémentaires de leur part. Cependant, une bonne planification de l'itération à concevoir évite d'imposer une pression indue aux programmeurs.

3.8. Client sur place

Dans MAIA, comme dans XP, le client a une place de choix au sein de l'équipe de projet. Dans un monde idéal, le client est physiquement près des programmeurs et peut répondre rapidement aux questions de ceux-ci, or ce n'est pas possible dans les projets typiques d'entrepôt. Les clients sont souvent des cadres supérieurs de l'entreprise, car les projets visent souvent la direction; il est impensable de les monopoliser durant un projet. [14]

MAIA propose une alternative : l'utilisation d'experts techniques pour remplacer le client final dans l'équipe. Un expert technique peut aider l'équipe TI avec ses connaissances dans le domaine d'affaires. [20] C'est la responsabilité de l'expert technique d'interpréter les besoins du client pour l'équipe TI.

Le client n'a peut être pas besoin d'être constamment sur place. Dans une étude de cas, il a été démontré qu'un projet peut être réussi avec une implication réelle du client durant 21 % du projet. [1] Pour que cette implication soit bénéfique pour l'équipe TI, le client doit être disponible au moment requis, pour conserver l'élément d'agilité de l'équipe TI.

Lorsque le client n'est pas constamment sur place, la stratégie de communication est importante. Dans une autre étude de cas, une corrélation a été établie entre la richesse de la communication entre le client et l'équipe TI et le nombre de défectuosités. [23] En effet, plus la communication est pauvre, plus le nombre de défectuosités est grand. La forme de communication la plus riche est la rencontre face à face, suivie par la téléconférence. Le courriel est la forme la plus pauvre de communication.

Les moments clés où le client devrait être présent physiquement sont :

- au début de l'itération, pour bien définir ce qui est important,
- lors de la définition des dimensions conformes, où l'équipe TI doit savoir exactement ce que le client veut,
- lors de l'élaboration des rapports, car le client peut apporter une rétroaction rapide pour avoir exactement ce dont il a besoin en fin d'itération et [32]
- lors des tests d'intégration.

Il est très important, dans les interactions avec le client, de bien l'éduquer sur les besoins de l'équipe TI et sur l'importance de sa présence pour le succès du projet. Sans une rétroaction suffisante, le client n'aura pas ce qu'il désire.

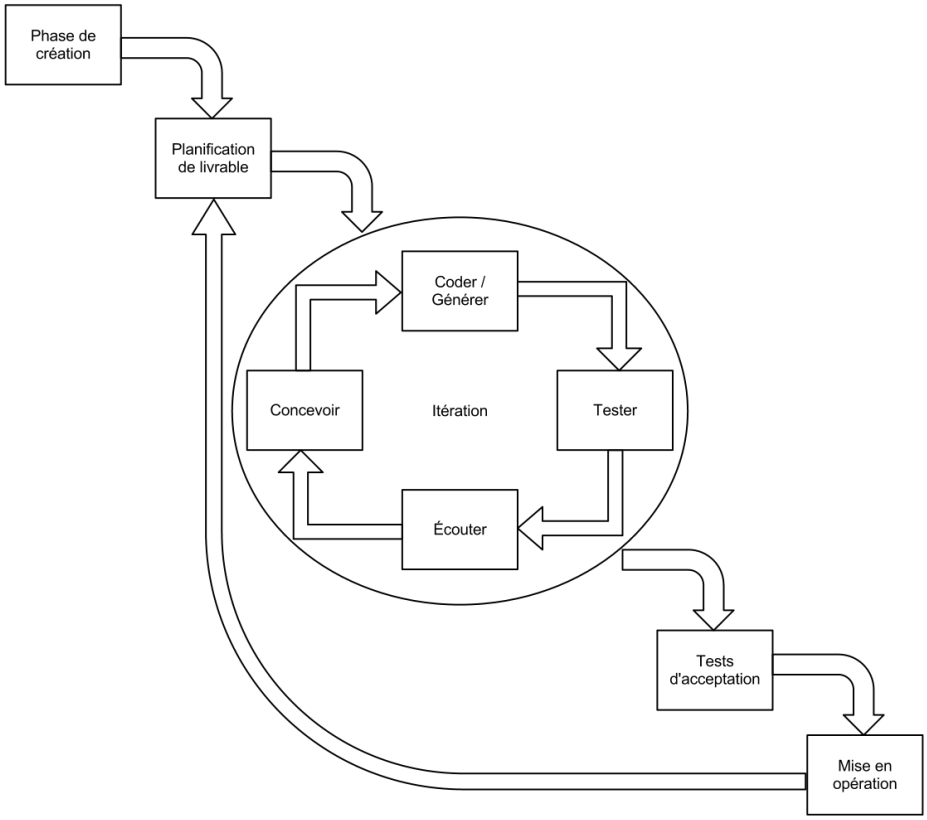

Figure 3.1 Étapes d'un projet avec MAIA

3.9. Cycle de vie d'un projet MAIA

Un projet utilisant MAIA passe par différentes étapes, tel qu'illustré dans la figure 3.1. Les prochaines sections définissent chacune des étapes. De plus, certains éléments spécifiques qui doivent être parties intégrantes d'un projet d'entrepôt sont décrits.

Avant de définir chacune des étapes du cycle de vie, il est important de déterminer l'importance des métadonnées dans un projet d'entrepôt et donc dans MAIA. Les métadonnées sont les données concernant les données de l'entrepôt.

Voici quelques exemples de métadonnées :

- la provenance de la donnée dans le système source,
- les transformations effectuées sur la donnée,
- les règles d'affaires régissant la donnée,
- la date et l'heure de la dernière extraction,
- l'état de la qualité de la donnée,
- la définition de la donnée et
- celles qui aident les outils de présentation tels la précision numérique et les masques d'édition.

Les métadonnées doivent répondre aux grandes questions des utilisateurs : [11]

- Qu'est-ce qui est disponible?
- Qu'est-ce que ça veut dire?
- Où est-ce que c'est?
- Comment est-ce arrivé là?
- Comment puis-je l'utiliser?

Le but d'une bonne gestion des métadonnées est de s'assurer qu'elle est correcte, complète, tenue à jour et accessible aux utilisateurs. [41] C'est pourquoi les métadonnées doivent être créées et maintenues durant toutes les étapes d'un projet d'entrepôt.

Les métadonnées peuvent influencer l'attitude des utilisateurs envers les entrepôts. En effet, les métadonnées peuvent rendre l'entrepôt plus convivial et plus facile à manipuler. Une dose adéquate de métadonnées augmenterait l'utilisation de l'entrepôt, mesure sans équivoque de

succès d'un projet. L'influence de la perception de l'utilité de l'entrepôt à travers les métadonnées est sensiblement la même que la qualité des données ou la qualité de la formation. [11]

Dans un projet avec MAIA, le strict minimum de documentation requis est la métadonnée. Une itération ne peut être livrée sans son ensemble de métadonnées.

Une certaine variété de métadonnées est générée lors de l'exécution des ETC et peut inclure la quantité d'enregistrements extraits ou la date de dernière extraction. La métadonnée générée peut aussi servir pour communiquer avec le responsable des données à propos des erreurs survenues lors de l'exécution des ETC.

Malgré les efforts des programmeurs pour prévoir tous les cas de figures possibles, il peut se glisser dans les données sources des cas qui ne respectent pas les règles d'affaires. Ce peut être dû à une coquille dans le système opérationnel ou un utilisateur qui utilise le système d'une manière fortuite. De plus, il y a des sources de données qui sont intrinsèquement non fiables, où les erreurs de données sont fréquentes et inévitables.

Peu importe la provenance de l'irrégularité, une action doit être prise. C'est pourquoi il doit y avoir un mécanisme clair pour détecter et rapporter les erreurs de données. Il est aussi important de bien planifier la métadonnée que le processus de gestion des erreurs.

Les éléments précédents sont essentiels à l'ensemble de la méthodologie. Les points suivants décrivent les étapes à suivre dans la méthodologie. La première étape dans un projet d'entrepôt est stratégique. Dans cette étape, l'étape de création, l'équipe TI doit investiguer, faire des recherches, discuter et prendre certaines décisions sur les grandes orientations du projet. Il faut commencer un projet avec une idée générale de la direction que prendra celui-ci, sinon un échec est probable. [21]

L'utilisation de la phase d'analyse de haut niveau d'Agile Modeling, tel que décrite dans la section 2.6, est recommandée pour aider l'équipe TI dans la planification en comptabilisant les

besoins des utilisateurs. C'est à la fin de la phase de création que la décision d'aller de l'avant avec le projet est prise.

Avec l'analyse de haut niveau de la phase de création qui donne une idée générale des besoins utilisateurs, une planification de l'itération à produire peut être effectuée. Avant de commencer, une évaluation de la capacité de production de l'équipe doit être réalisée car elle déterminera quelle quantité de fonctionnalités fera partie de la prochaine itération. En utilisant le jeu de la planification avec le client, le contenu de l'itération est défini et l'équipe TI peut commencer à concevoir.

C'est à cette étape qu'est décidé si l'itération ira en production ou sera mise en attente après les tests d'acceptation, pour une livraison de plusieurs itérations en même temps. Tout dépend de la culture de l'entreprise face aux installations.

La prochaine phase est l'exécution de l'itération. C'est à ce moment que le produit est conçu. Voici la liste des activités de MAIA, inspirée de XP (voir la figure 2.1) :

- concevoir,
- programmer ou générer,
- tester et
- écouter.

Deux programmeurs travaillent en paires dès la conception, tel que décrit dans la section 3.5. Dans cette étape, les programmeurs auront plusieurs opérations à faire :

- analyse des besoins détaillés avec le client,
- analyse des données à la source,
- analyse du modèle dimensionnel,

- cartographie de la source vers la cible,
- détermination de la stratégie de gestion des erreurs,
- prototype des rapports et tableaux de bord et
- définition des tests à effectuer.

Après la conception, les programmeurs implémentent ce qu'ils viennent de concevoir. Le terme programmation de cette étape est pris dans un sens très large. Dans un entrepôt, il y a de la configuration d'outil, de la création de rapports ou de tableaux de bord, de la mise en place de cubes OLAP et parfois même de la programmation d'interfaces graphiques. Il y a aussi la création logique et physique de schémas de base de données. Cela inclut les tests unitaires automatisés.

Lorsque la génération est terminée, les programmeurs testent les fonctionnalités élaborées à l'aide des tests unitaires automatisés et d'autres tests qui doivent être exécutés manuellement. À cette étape, la collaboration du client est essentielle pour obtenir une rétroaction sur la fonctionnalité implémentée.

Enfin, il faut écouter le client. C'est le client qui confirme que la fonctionnalité sur laquelle le programmeur travaille est complète et comble son besoin. Il doit déterminer la prochaine fonctionnalité à concevoir, parmi la liste définie dans la phase de planification et peut ajouter ou retirer des fonctionnalités non-implémentées de la liste.

Cette boucle est répétée jusqu'à ce que toutes les fonctionnalités prévues dans l'itération soient générées, testées et approuvées.

L'itération est complète et testée unitairement. La phase de tests d'acceptation couvre les tests fonctionnels, d'acceptation et de régression. Ce n'est pas seulement l'itération qui est testée, mais tout ce qui est susceptible d'être influencé par l'itération. Ce pourrait être, dans les cas les plus complexes, l'entrepôt au complet.

Les tests d'acceptation sont sous la responsabilité du département d'assurance qualité, en collaboration avec le client. Durant cette phase, les programmeurs participent aux tests et restent disponibles pour toute correction nécessaire.

Dans la phase de mise en opération, les programmeurs préparent le déploiement de l'itération pour les responsables dans l'entreprise pour les mises en opération. De plus, un transfert de connaissances doit être fait vers les équipes de support pour les fonctionnalités ajoutées.

3.10. Applicabilité de la méthodologie MAIA

La méthodologie Agile d'intelligence d'affaires est basée sur les pratiques et activités reconnues de XP avec des étapes venant de la méthodologie Kimball. Elle est itérative et incrémentale tout en étant adaptée à des projets d'entrepôt. En adoptant des pratiques Agiles, elle est en mesure de réduire les risques connus dans les projets traditionnels d'entrepôt.

Comme la plupart des méthodologies agiles, MAIA n'est pas une solution pour tous les projets. Certains critères doivent être présents pour espérer un succès avec cette méthodologie.

Du côté technologique, il est plus facile d'utiliser MAIA lorsque les outils, plateformes et autres artifices technologiques sont en place. Dans l'éventualité que la découverte et la mise en place des technologies fassent partie du projet, il est recommandé d'isoler dans une ou plusieurs itérations cette activité du reste du développement. Ce peut être la phase zéro d'un projet. Le niveau d'agilité de cette phase dépend beaucoup du contexte de l'entreprise et de la ségrégation des tâches.

Du côté des données, MAIA est une approche spécialement performante lorsqu'une portion de l'entrepôt est déjà livrée. Ross, du groupe Kimball, appuie les techniques Agiles dans l'entrepôt lorsque les dimensions sont conformes, accélérant la conception. [34]

L'implication du client ou de son représentant influence le choix de méthodologie. Un client qui ne veut pas être impliqué dans le projet, qui croit que faire un projet d'entrepôt est comme

se procurer une automobile, ne participera pas dans une méthodologie telle que MAIA; pour un tel client, une approche traditionnelle est plus appropriée.

Enfin, il faut que l'équipe TI soit prête à modifier ses manières de faire et embrasser les pratiques agiles proposées dans cet essai. Il faut donc prendre en considération que ce n'est pas donné à toutes les personnes de vouloir travailler en paires tout le long d'un projet ou de concevoir les tests avant d'avoir créé sa partie du projet. Il est prudent d'introduire les aspects de MAIA un par un, de même que évaluer le niveau de confort des membres de l'équipe et l'efficacité de chaque aspect. Il est possible que certaines pratiques ne conviennent pas au style de l'entreprise ou ses employés; ces pratiques peuvent être éliminées sans éliminer les gains des pratiques mises en place.

Conclusion

Les méthodologies traditionnelles pour la gestion d'un projet TI d'entrepôt ont un taux d'échec élevé, qui peut varier de 41 % à 90 %. Selon Trembly, [42] les principales causes d'échec de ces projets sont attribuables à une incompréhension des besoins du client de la part des TI, et la qualité du produit final.

Dans les projets TI de développement logiciel, les méthodologies Agiles permettent d'adresser ces facteurs, et leurs pratiques réduisent le risque d'échec.

La méthodologie MAIA proposée par cet essai emprunte les pratiques du jeu de la planification et des petits livrables de XP et *Scrum,* et les applique à l'univers des projets d'entrepôt pour permettre au client d'avoir plus rapidement ce qu'il désire, en lui offrant la chance d'exercer une rétroaction constructive. Cette rétroaction permet de réduire l'incompréhension des TI chez le client, car ce dernier peut alors mieux articuler ses besoins à travers ses commentaires du produit concret. Cette rétroaction permet également aux TI de mieux interpréter le client, car les commentaires du client valide la compréhension des besoins par TI.

De plus, MAIA emprunte aussi de XP les pratiques de tests, de réusinage, de programmation en paires et d'intégration continue pour augmenter la qualité de l'entrepôt, autant dans la justesse des données et l'efficacité des outils d'exploitation que dans les éléments en arrière-scène tels les ETC.

L'importance des métadonnées dans un projet d'entrepôt a été soulignée. Leur qualité est essentielle pour bien transmettre au client les informations que l'entrepôt contient et comment les utiliser.

Un autre aspect propre aux entrepôts est la saine gestion des problèmes de qualité des données. Les utilisateurs continueront à exploiter les entrepôts tant qu'ils auront confiance en la qualité

de leurs données. Sans une constante surveillance de cette qualité, l'entrepôt peut induire les utilisateurs en erreur avec des données incorrectes.

MAIA n'est pas une solution miracle. Elle ne peut pas s'appliquer dans n'importe quel projet d'entrepôt. Son application est cependant idéale s'il y a déjà un ensemble de dimensions conformes pour accélérer le développement. MAIA est donc recommandée pour un projet d'entrepôt où le client a qu'une idée vague de ses besoins. Une équipe TI peut utiliser MAIA si elle est relativement petite et s'estime prête à utiliser des pratiques Agiles pour augmenter ses chances de réussite.

Cet essai propose une méthodologie intégrant les idéaux du mouvement Agile et certains éléments propres au monde de l'intelligence d'affaires. Deux opportunités découlent de cette proposition.

La première opportunité se veut le raffinement de la méthodologie avec d'autres pratiques, telles le prototypage. En effet, il est possible d'utiliser des prototypes, faits par exemple dans un chiffrier électronique, pour communiquer avec le client et recevoir rapidement une rétroaction. Une telle technique permettrait de déterminer une volumétrie et clarifier les règles d'affaires avant de débuter la programmation des ETC.

La seconde opportunité est d'appliquer la méthodologie MAIA dans un contexte réel et en évaluer les résultats. En utilisant cette méthodologie, sa valeur pourra en être déterminée et MAIA pourra évoluer.

Enfin, il est possible que la solution ultime soit de repenser complètement l'approche, et de considérer un projet d'entrepôt aussi différent des autres projets TI qu'un projet de maçonnerie est différent d'un projet de menuiserie. En effet, un des aspects les plus importants dans un entrepôt et son seul produit est la donnée de qualité. Au lieu d'aligner un projet d'entrepôt sur un paradigme tel que *Lean*, qui provient du Toyota Production System, pourquoi ne pas utiliser un différent paradigme, tel que Six Sigma, pour mettre l'accent sur la qualité totale des données et ainsi assurer aux clients qu'ils peuvent se fier complètement sur les connaissances

dérivées de l'entrepôt pour prendre des décisions qui peuvent changer l'orientation de leur compagnie.

Les programmeurs objets ont fait évoluer leurs méthodologies depuis longtemps, il est grand temps que les développeurs d'entrepôt emboîtent le pas.

Références

[1] Abrahamsson, P. et Koskela, J., « Extreme programming : a survey of empirical data from a controlled case study », *International Symposium on Empirical Software Engineering*, 19 août 2004, pp. 73-82.

[2] Ambler, S. W. et Lines, M., *Disciplined Agile Delivery : A Practitioner's Guide to Agile Software Delivery in the Enterprise*, 1re éd., 23 mai 2012, 544 p.

[3] Beck, K., « Manifeste pour le développement Agile de logiciels », http://agilemanifesto.org/iso/fr, page consultée le 2 juillet 2012.

[4] Beck, K., *Extreme Programming Explained: Embrace Change*, 1re éd., USA, 2000, 203 p.

[5] Breur, T., « Data quality is everyone's business – designing quality into your data warehouse - part 1 », *Journal of Direct, Data and Digital Marketing Practice*, vol. 11, nº 1, 14 avril 2009, pp. 20-29.

[6] Breslin, M., « Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models », *Business Intelligence Journal*, vol. 9, no 1, 2004, pp. 6-20.

[7] Brooks, F., *Mythical Man-Month: essays on software engineering*, Anniversary ed., USA, 1975, 322 p.

[8] Connor, D., « Report: Data Warehouse Failures Commonplace », *Network World*, vol. 20, no 3, 20 janvier 2003, p. 24.

[9] Cutter Consortium, « 41 % Have Experienced Data Warehouse Project Failures », http://www.cutter.com/research/2003/edge030218.html, page consultée le 7 août 2012.

[10] Foley, J., « Data Warehouse Pitfalls », *Information Week*, nº 631, 19 mai 1997, pp. 93-96.

[11] Foshay, N., Mukherjee, A. et Taylor, A., « Does Data Warehouse End-User Metadata Add Value », *Communications of the ACM*, vol. 50, nº 11, novembre 2007, pp. 70-77.

[12] Fowler, M., « The New Methodology », http://www.martinfowler.com/articles/newMethodology.html, 19 juin 2012.

[13] Gharaibeh, N., Abu-Soud, S. M., Bdour, W. et Gharaibeh, I., « Agile Development Methodologies: Are they suitable for developing Decision Support Systems », *Second International Conference on the Applications of Digital Information and Web Technologies*, 2009, pp. . 84-89.

[14] Goede, R. et Huisman, M., « The Suitability of Agile Systems Development Methodologies for Data Warehouse Development », *International Conference on Information Management Evaluation*, 2010, pp. . 99-106.

[15] Golfarelli, M., Rizzi, S., « Data Warehouse Testing », *International Journal of Data Warehousing and Mining*, vol. 7, n° 2, 2011, pp. 26-43.

[16] Highsmith, J., « History: The Agile Manifesto », http://agilemanifesto.org/history.html, page consultée le 2 juillet 2012.

[17] Holck, J. et Jørgensen, N., « Continuous Integration and Quality Assurance: a Case Study of Two Open Source Projects », *Australasian Journal of Information Systems*, Special Issue 2003/2004, 2003, pp. . 40-53.

[18] Inmon, W. H., *Building the data warehouse*, 4 éd., Wiley, New York, 2005, 543 p.

[19] Jukic, N. et Velasco, M., « Data warehousing requirements collection and definition : Analysis of a failure », *International Journal of Business Intelligence Research*, vol. 3, nº 1, 2010, pp. 66–76.

[20] Jyothi, V. E. and Rao, K. N., « Effective Implementation of Agile Practices », *International Journal of Advanced Computer Science and Applications*, Vol. 2, No. 3, 2011, pp. 41-48.

[21] Kimball, R., « Design Tip #111 Is Agile Enterprise Data Warehousing an Oxymoron », http://www.kimballgroup.com/2009/04/01/design-tip-111-is-agile-enterprise-data-warehousing-an-oxymoron/, page consultée le 6 août 2012.

[22] Kimball, R. et Ross, M., *The data warehouse toolkit*, New York, 1998, 436 p.

[23] Koskela, J., Abrahamsson, P. et Kyllōnen, P., « On-Site Customer in an XP Project: Empirical Results from a Case Study », *Lecture Notes in Computer Science*, Vol. 3281, 2004, pp. 1-11

[24] Larson, D., « BI principles for agile development: Keeping focused », *Business Intelligence Journal*, vol. 14, no 4, 2009, pp. 36-41.

[25] Mann, C. et Maurer, F., « A case study on the impact of scrum on overtime and customer satisfaction », *Agile Conference*, 2005, pp. 70-79.

[26] McMahon, J., « 5 Lessons from Transitioning to eXtreme Programming: Using two person teams to write software code is one aspect of eXtreme Programming

(XP). Here's what Citect learned when it implemented the XP method », *Control Engineering*, vol. 50, no 3, mars 2003, pp. 59-60.

[27] Office québécois de la langue française, « Expert technique », http://www.gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=17487217, page consultée le 31 juillet 2012

[28] Office québécois de la langue française, « Technologies de l'information », http://www.gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8875723, page consultée le 4 janvier 2012.

[29] Pierce, D., « Extreme Programming », *The Computer Bulletin*, vol. 44, nº 3, mai 2002, p. 28.

[30] Poppendieck, M. et Poppendieck T., *Implementing Lean Software Developpement : From Concept To Cash*, 1 éd., USA, 2006, 304 p.

[31] Race, P., « Using feedback to help students to learn », *The Higher Education Academy*, 2001, pp. 1-11.

[32] Rehani, B., « Agile way of BI implementation », *Annual IEEE India Conference (INDICON)*, 16 décembre 2011, pp. 1-6.

[33] Riggle, M., « Breaking the Cycle of Failure », *Intelligent Enterprise*, vol 4, no 12, 10 août 2001, pp. 40-45.

[34] Ross, M., « Design Tip # 135 Conformed Dimensions as the Foundation for Agile Data Warehousing », http://www.kimballgroup.com/html/11dt/DT135ConformedDimensionsFoundationAgileDW.pdf, page consultée le 6 août 2012.

[35] Schutte, S., Ariyachandra, T. et Frolick, M., « Test-Driven Development of Data Warehouses », *International Journal of Business Intelligence Research*, vol. 2, no 1, 2011, pp. 64-73.

[36] Schwaber, K., *Agile Project Management With Scrum*, 1 éd., USA, 2004, 192 p.

[37] Solomon, M. D., « Ensuring a Successful Data Warehouse Initiative », *Information Systems Management*, vol. 22, nº 1, 12 décembre 2005, pp. 26-36.

[38] Stapleton, J., « DSDM : Dynamic Systems Development Method », *Technology of Object-Oriented Languages and Systems*, 1999, p. 406.

[39] Spillner, A., Linz, T., Rossner, T. et Winter M., *Software Testing Practice: Test Management*, 1 éd, Rocky Nook, 24 août 2007, 339 p.

[40] TechRepublic, « Understanding the Pros and Cons of the Waterfall Model of Software Development », http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/6118423, page consultée le 8 septembre 2012.

[41] Thornthwaite, W., « Design Tip #75 Creating the Metadata Strategy », http://www.kimballgroup.com/2006/01/13/design-tip-75-creating-the-metadata-strategy/, page consultée le 6 août 2012.

[42] Trembly, A. C., « Experts: Technology is not to Blame for Data Warehouse Failures », *National Underwriter*, vol. 105, nº 45, 5 novembre 2001, pp. 32-41.

[43] Wikipédia, « Dynamic Systems Development Method », http://en.wikipedia.org/wiki/Dynamic_systems_development_method, page consultée le 22 juillet 2012.

[44] Wikipedia, « Waterfall Model », http://en.wikipedia.org/wiki/Waterfall_model, page consultée le 30 octobre 2012.

[45] Williams L. et Erdogmus H., « On the Economic Feasibility of Pair Programming », *International Workshop on Economics-Driven Software Engineering in conjunction with the International Conference on Software Engineering*, 2002, pp. 1-7.

[46] Zhai, L., Hong, L. et Sun, Q., « Research on Requirement for High-quality Model of Extreme Programming », *2011 International Conference on Information Management (ICIII)*, 2011, pp. 518-522.

Annexe 1

Tableaux détaillés des méthodologies

Tableau A1.1 Liste des principes des méthodologies Agiles

Principes des méthodologies Agiles
Satisfaire le client à travers la livraison en continu de logiciels de valeur.
Recevoir les demandes de changement, même si elles sont reçues tard dans la conception. Les techniques agiles exploitent les changements pour donner l'avantage compétitif au client.
Livrer un logiciel fonctionnel fréquemment, de quelques semaines à quelques mois, de préférence à la fréquence la plus courte possible.
Les représentants de l'entreprise et les développeurs travaillent ensemble au jour le jour durant le projet.
Bâtir des projets avec des gens motivés : donnez-leur un environnement et le support dont ils ont besoin et faites-leur confiance.
La meilleure façon de transmettre l'information est la conversation face à face.
Le logiciel fonctionnel est la mesure de progrès d'un projet.
Les méthodologies Agiles font la promotion du développement durable (on peut garder la cadence indéfiniment).
Une attention continue à l'excellence technique aide à rester agile.
La simplicité est essentielle; il est plus facile d'ajouter à un processus trop simple que d'en enlever à un processus trop compliqué.
Les meilleurs architectures, requis et designs viennent d'équipes auto-organisées.
L'équipe Agile améliore constamment ses processus Agiles.

Traduction libre

Source : Beck, K. (2001), p. 1

Tableau A1.2 Liste des formes de gaspillage selon la méthodologie Lean

Les sept formes de gaspillage selon Lean	
Travail partiellement fait	Une fonctionnalité qui n'est pas

Les sept formes de gaspillage selon Lean	
	complétée peut se perdre, avoir des défauts et devenir obsolète.
Fonctionnalités supplémentaires	La plus grande source de gaspillage dans le développement de logiciel est la présence de fonctionnalités supplémentaires. Selon Poppendieck, il peut y avoir jusqu'à deux tiers des fonctionnalités ne sont pas utilisées, ou rarement. La présence de ces fonctionnalités complique la maintenance, hausse les coûts et réduit la vie utile du logiciel. [30] Dans un projet d'entreposage, ce pourrait être des dimensions ou mesures superflues qui ne font que gonfler l'entrepôt pour en réduire sa performance.
Réapprentissage	Devoir réapprendre quelque chose qui a déjà été appris, mais oublié.
Transferts de dossiers	Le transfert d'un dossier d'une personne à l'autre constitue une perte d'information, car le savoir tacite ne peut jamais être complètement transféré.
Commutation de tâches	L'exécution simultanée de plusieurs tâches par la même personne demande plus de temps que les faire en séquence.

Les sept formes de gaspillage selon Lean	
	En effet, ce type d'exécution entraîne des heures supplémentaires requises par le changement de tâche.
Délais	Du temps est perdu à chaque fois qu'un programmeur doit attendre une décision.
Défauts	Un défaut dans le logiciel est un gaspillage, car un effort supplémentaire sera requis pour régler le problème. Un défaut d'un entrepôt peut être un manque de qualité des données pouvant entraîner des pertes monétaires si des décisions d'affaires sont prises en l'utilisant.

Traduction libre

Source : Poppendieck, M. (2006), p. 47

Tableau A1.3 Liste des pratiques de XP

Pratiques de XP
Le jeu de la planification
Petits livrables
Métaphore
Conception simple
Tester
Réusinage
Programmation en paires
La communauté est propriétaire du code
Intégration continue
Semaine de quarante heures
Client sur place
Standards de programmation

Traduction libre

Source : Beck, K. (2001), p. 1