View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CorpusUL

RICHARD TREMBLAY

IMPLANTATION D'UNE MÉTHODE AGILE DE DÉVELOPPEMENT LOGICIEL EN ENTREPRISE

Une culture accueillant le changement

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de maître (M. Sc.)

DÉPARTEMENT D'INFORMATIQUE ET DE GÉNIE LOGICIEL
FACULTÉ DES SCIENCES ET GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2007

© Richard Tremblay, 2007

Résumé

Depuis quelques années, les méthodes agiles ont émergées et semblent prometteuses. Ce mémoire relate les travaux réalisés dans le but de procéder à l'implantation d'une méthode agile de développement en entreprise. Afin de distinguer les approches agiles, nous débutons par un rappel des approches traditionnelles. Nous établissons quelles sont les caractéristiques de ces approches, les différents modèles et leurs limitations. Nous analysons ensuite l'offre des approches agiles. Nous définissons en quoi consiste l'agilité et quelles sont les caractéristiques communes de ces approches. Nous présentons quelques méthodes, plus particulièrement : Extreme Programming, Scrum et Crystal Clear. Finalement, nous relatons l'expérience d'une implantation en entreprise afin de vérifier sa facilité d'application. Nous concluons que ces approches adaptatives sont plus efficaces que les approches prédictives lorsqu'elles sont utilisées dans un contexte propice.

Abstract

In recent years, agile methods have emerged and appear promising. This memoirs describes the work carried out in order to proceed with the implementation of an agile method development in business. To distinguish the agile approaches, we begin with a recap of traditional approaches. We establish what are the characteristics of these approaches, the various models and their limitations. We then analyze the proposal of agile approaches. We define what constitutes agility and what are the common characteristics of these approaches. We present some methods, in particular: Extreme Programming, Scrum and Crystal Clear. Finally, we talk about the experience of an establishment of the agile method into a business in order to verify its ease of implementation. We conclude that these adaptive approaches are more effective then predictive approaches when used in the right context.

Avant-propos

Je remercie mon directeur, Nadir Belkhiter, qui a cru en moi et à ma persévérance. Ce projet a duré plusieurs années, auxquelles il m'était difficile d'investir tout le temps requis pour faire avancer rapidement ce projet.

Je remercie également mon équipe de travail et mon employeur. Ils ont été au centre de mes expérimentations. Leur ouverture d'esprit et l'autonomie dont je disposais m'ont permis de réaliser ce projet.

Un merci particulier à Louis Bastarache qui m'orienta et me rassura sur mes points de vue. Professionnellement, je n'ai jamais eu la chance de travailler avec un senior qui m'aurait montré la voie et fait bénéficier de son expérience. Ce projet m'a permis de bénéficier d'un tel appui.

Finalement, la famille reste au cœur des gens qu'il faut remercier, en débutant par mon père. Étant l'aîné d'une famille de six enfants, il n'a pas eu la chance de poursuivre ses études, mais il m'a fait rapidement comprendre que l'éducation est capitale et que nous devons persévérer pour atteindre nos objectifs dans la vie. Je remercie également ma conjointe et nos deux enfants avec qui j'ai partagé mon temps entre la vie de famille et le temps investi pour la recherche et la rédaction de ce mémoire.

Je vous dis merci à tous.

À mon père

Table des matières

Chapitre 1: Les approches traditionnelles....5
1.1 Origine des approches traditionnelles et du génie logiciel....5
1.1.1 Le domaine du génie logiciel....6
1.2 Les processus de production dans une approche traditionnelle....7
1.2.1 La gestion de projets....7
1.2.2 Organisation du travail....8
1.2.3 Les modèles reconnus....11
1.2.4 Adaptations des modèles....13
1.2.5 Méthodes populaires....13
1.2.6 Validation du processus....14
1.3 Le produit dans une approche traditionnelle....14
1.3.1 Spécifications du produit....15
1.3.2 La qualité du produit....15
1.3.3 La standardisation....16
1.4 Les développeurs dans une approche traditionnelle....16
1.4.1 Les rôles et responsabilités....16
1.4.2 Les outils logiciels....17
1.5 Le client dans une approche traditionnelle....19
1.6 Analyse des approches traditionnelles....20
1.6.1 Application favorable des approches traditionnelles....20
1.6.2 Limitation des approches traditionnelles....20
1.7 Conclusion....21
Chapitre 2: Caractéristiques des méthodes agiles....23
2.1 Les bases de l'agilité....23
2.1.1 Définition du terme « Agile »....23
2.1.2 Origines des approches agiles....24
2.1.3 Le manifeste agile....25
2.1.4 Les principes agiles....27
2.2 L'offre agile....29
2.2.1 Perceptions initiales....29
2.2.2 L'apport des approches agiles....30
2.2.2.1 L'apport agile pour les développeurs....31
2.2.2.2 L'apport agile pour le produit....32
2.2.2.3 L'apport agile pour le client....33
2.2.2.4 L'apport agile pour la gestion de projet....34
2.3 Position de l'agilité....35
2.3.1 Conformité aux modèles certifiés....35
2.4 Conformité au modèle agile....37
2.4.1 Conformité agile du RAD....37
2.4.2 Analyse de RUP....38
2.4.3 Analyse des développements libres....38
2.4.4 Limitations des approches agiles....39
2.4.4.1 Limitations liées au type de projet....39
2.4.4.2 Limitations liées à la culture organisationnelle....40

2.4.5 Les supports agiles 41
2.4.5.1 La communauté 41
2.4.5.2 Les outils 41
2.4.5.3 Adoption par l'industrie 42
2.5 Distinction entre les approches traditionnelles et agiles 43
2.5.1 Propriétés communes et spécifiques 43
2.6 Conclusion 47
Chapitre 3: Description de méthodes agiles 49
3.1 Extreme Programming 49
3.1.1 Origines de XP 49
3.1.1.1 Son inventeur 49
3.1.1.2 Les racines de XP 50
3.1.2 Description de la méthode 50
3.1.2.1 Les valeurs de XP 50
3.1.2.2 Les pratiques 51
3.1.2.3 Processus d'extreme Programming 52
3.1.3 Conclusion sur XP 60
3.2 SCRUM 60
3.2.1 Origines de SCRUM 60
3.2.1.1 Son inventeur 60
3.2.1.2 Évolution de ADM vers SCRUM 61
3.2.1.3 Une nouvelle joute de développement 61
3.2.1.4 Les types de processus 62
3.2.1.5 L'amalgame de SCRUM 63
3.2.2 Description de la méthode 64
3.2.2.1 Vocabulaire de SCRUM 64
3.2.2.2 Les trois phases de SCRUM: 65
3.2.2.3 Les contrôles 68
3.3 Famille Crystal 69
3.3.1 Origine de Crystal 69
3.3.1.1 Son inventeur 69
3.3.1.2 La famille de méthode Crystal 70
3.3.2 Description de la méthode 71
3.3.2.1 Les sept propriétés de Crystal Clear 71
3.3.2.2 Les stratégies et les techniques 74
3.3.2.3 Cycle de développement 78
3.3.3 Conclusion sur Crystal Clear 81
3.4 Autres méthodes agiles 82
3.4.1 Lean Software Development 82
3.4.2 Dynamic System Development Method 84
3.4.2.1 Cycle de développement de DSDM 85
3.4.3 Feature Driven Development 87
3.4.3.1 Cycle de développement de FDD 88
3.4.4 Adaptive System Development 90
3.4.4.1 Cycle de développement de ASD 92

Chapitre 4: Implantation d'une approche agile 95
4.1 Contexte d'origine 95
4.1.1 Description de l'équipe à l'origine 95
4.1.2 Description du produit à l'origine....................96
4.1.3 Description du processus à l'origine....................96
4.1.4 Description de la clientèle à l'origine 97
4.2 Analyse du contexte et objectifs d'amélioration....................97
4.2.1 Analyse du contexte....................98
4.2.1.1 Analyse du contexte de l'équipe....................98
4.2.1.2 Analyse du contexte du produit 98
4.2.1.3 Analyse du contexte du processus 99
4.2.1.4 Analyse du contexte de la clientèle....................101
4.2.2 Objectifs d'amélioration 102
4.2.3 Méthodes candidates....................103
4.2.3.1 Adéquation de extreme Programming 103
4.2.3.2 Adéquation de Scrum....................104
4.2.3.3 Adéquation de Crystal Clear....................104
4.2.3.4 Commentaire sur l'implantation 105
4.3 Gestion du changement....................105
4.3.1 Démarche d'implantation 105
4.3.1.1 Présenter l'urgence d'intervenir....................106
4.3.1.2 Composer une coalition pour mener le changement....................106
4.3.1.3 Créer une vision pour guider les efforts....................106
4.3.1.4 Communiquer la vision....................106
4.3.1.5 Impliquer les gens et les inciter à intervenir 107
4.3.1.6 Planifier pour créer de petites victoires 107
4.3.1.7 Consolider les améliorations et engendrer d'autres changements 107
4.3.1.8 Institutionnaliser la nouvelle approche 107
4.3.2 Chronologie des événements 107
4.4 Analyse des résultats obtenus 110
4.4.1 Changements apportés à l'équipe 110
4.4.2 Changements apportés au produit....................111
4.4.3 Changements apportés au processus....................112
4.4.4 Changements apportés avec nos clients....................114
4.5 Atteinte de l'agilité 114
4.6 Conclusion 115
Chapitre 5: Conclusion 118
5.1 L'offre traditionnelle....................118
5.2 L'offre agile 118
5.3 Constatations sur l'agilité 119
5.4 Questionnement initial....................121

Liste des tableaux

Tableau 1-1 : Description des tâches du cycle de vie logiciel 10
Tableau 1-2 : Principaux titres et rôles des approches traditionnelles.................................17
Tableau 2-3 : Le manifeste agile...25
Tableau 2-4 : Caractéristiques des équipes de développement..44
Tableau 2-5 : Caractéristiques liées au produit...44
Tableau 2-6 : Caractéristiques de la gestion de projet 45
Tableau 2-7 : Caractéristiques de l'organisation 46
Tableau 3-8 : Domaines de gaspillage appliqués au logiciel...83
Tableau 4-9 : Propriétés, stratégies et techniques de Crystal Clear 105
Tableau 5-10 : Métriques agiles de Crystal 145

Liste des figures

Figure 1-1 : Répartition des efforts de développement....6
Figure 2-2 : Spectre de la planification....36
Figure 2-3 : Continuum adaptatif à prédictif....43
Figure 3-4 : Processus XP au niveau du projet....53
Figure 3-5 : Processus XP au niveau de l'itération....54
Figure 3-6 : Processus XP au niveau du développement....55
Figure 3-7 : Processus XP sur la propriété du code commun....56
Figure 3-8 : Progression des tests acceptés par itérations....58
Figure 3-9 : Séquence des activités selon différentes méthodologies....58
Figure 3-10 : Apprentissage de l'estimation de la valeur et des coûts....59
Figure 3-11 : Probabilité de succès en fonction de la complexité du projet....63
Figure 3-12 : Processus global de SCRUM....65
Figure 3-13 : Matrice de la famille Crystal....71
Figure 3-14 : Imbrication des différents cycles de Crystal....79
Figure 3-15 : Pyramide inversée de DSDM....84
Figure 3-16: Cycle de DSDM....85
Figure 3-17: Cycle de développement de FDD....88
Figure 3-18 : Les grands thèmes et les étapes de ASD....92
Figure 5-19 : XP vue en couche d'oignon....136
Figure 5-20 : SCRUM en une page de Wake....141
Figure 5-21 : Modèle des niveaux du CMMi....149

Introduction

Une facette du domaine du génie logiciel se spécialise dans l'étude de la démarche utilisée pour la création du logiciel. Les approches de développement se concrétisent par des méthodes. Depuis le milieu des années 1990, une nouvelle génération de méthodes a vu le jour. Les méthodes agiles proposent une vision différente du développement logiciel qui est présentée dans ce mémoire.

Objectif

L'objectif est de réaliser l'implantation d'une méthode agile en entreprise, afin de témoigner des efforts nécessaires concernant son application dans la réalité. Je désire aussi vérifier l'efficacité des méthodes agiles, afin de déterminer si ces méthodes apportent un changement dans la conception et le développement du logiciel.

Par la réalisation de ce projet, je cherche à rassurer ou mettre en garde les équipes qui aimeraient implanter une de ces méthodes. Au terme de cette expérimentation, j'estime pouvoir répondre aux questions suivantes:

- Le peu de documentation générée par ces méthodes, est-elle suffisante pour l'apprentissage du système par les nouveaux utilisateurs ?
- Les équipes de développement peuvent changer et l'entretien peut aussi être assuré par des consultants externes. Auront-ils assez d'information pour pouvoir répondre rapidement et efficacement aux demandes du client ?
- Peut-on satisfaire aux exigences de la norme ISO-9001 : 2000 en utilisant une telle méthode?

Intérêt

Plusieurs organisations ou groupes qui développent des logiciels, ne connaissent que les approches traditionnelles qui sont souvent lourdes et coûteuses à gérer. On critique principalement le dépassement de coûts et les délais de livraison, ce qui amène les petites équipes à délaisser ces approches. À défaut, peu de méthodes alternatives sont présentées. Il y a un écart entre l'univers des praticiens et les modèles théoriques.

L'évolution des technologies et le manque de ressources sont des facteurs qui contribuent à la diminution de la grosseur des équipes. Il est important de trouver des approches adéquates à leur contexte organisationnel.

Les approches agiles offrent possiblement une opportunité afin de répondre au besoin d'une méthode suffisamment structurée. Elle doit permettre de mener des projets dans une zone sécuritaire. Il est préférable d'adopter une méthode qualifiée « d'assez bonne » qu'aucune méthode.

En s'appuyant sur la collaboration des intervenants, la communication et de petites livraisons, ces approches permettent de livrer un produit satisfaisant tout en respectant les budgets et les échéanciers.

Structure du mémoire

Le mémoire est composé de cinq chapitres, qui, dans l'ordre, présentent les approches traditionnelles, les approches agiles, les méthodes agiles, l'implantation et la conclusion.

Le chapitre 1 définit les éléments qui composent les approches traditionnelles. Il permet de faire un rappel sur les paradigmes actuels de ces approches. Il présente les éléments, suivant une structure qui facilite la comparaison avec les approches agiles du deuxième chapitre.

Le chapitre 2 présente les caractéristiques des approches agiles, tout en reprenant une structure similaire au premier chapitre. Il définit l'agilité en fonction de ses valeurs et de ses principes. Il fait état d'un bilan comparatif qui distingue les différents types d'approches.

Le chapitre 3 distingue les principales méthodes agiles. Il détaille particulièrement trois d'entre-elles: Extreme Programming, Scrum et Crystal Clear. Chaque section passe en revue les origines et les particularités de chacune des méthodes. Des annexes complètent certaines observations. Ce chapitre permet aussi un survol de plusieurs autres méthodes reconnues.

Le chapitre 4 présente l'implantation. Afin de juger les améliorations, on y décrit la situation initiale et les points d'améliorations recherchés. On y présente une stratégie d'implantation et le suivi des événements. Finalement, nous revenons sur les objectifs de départ et présentons le niveau d'atteinte des points d'amélioration visés.

Le chapitre 5 est la conclusion générale. Il résume les chapitres précédents et présente les constatations faites au cours de cette expérimentation.

Méthodologie

La méthodologie suivie était constituée de trois grandes phases : Analyser les approches reconnues et leurs limitations ; Approfondir mes connaissances des approches agiles, pour déterminer laquelle serait applicable ; Procéder à l'implantation de l'une d'entre elles, pour en vérifier l'efficacité et les efforts requis.

En un premier temps, il fallait analyser l'état général des approches traditionnelles, afin de retracer les démarches antérieures ayant mené aux paradigmes actuels.

Deuxièmement, il fallait répertorier les approches agiles par une revue de littérature, afin d'identifier les innovations qu'elles apportent.

Troisièmement, il fallait concevoir et appliquer une stratégie d'implantation de la méthode, afin d'analyser les résultats en dressant un bilan des changements avec la situation initiale.

Chapitre 1: Les approches traditionnelles

L'objectif de ce chapitre est de préparer une comparaison entre les approches de développement traditionnelles et agiles. Pour ce faire, un rappel des principaux éléments qui composent le génie logiciel est approprié. Le chapitre commence par une rétrospective historique qui retrace les événements ayant motivé l'industrie à structurer le domaine. Par la suite, les éléments sont regroupés sous les thèmes suivants : le processus de production, le produit, l'équipe de développement et le client. Ces mêmes thèmes se retrouvent dans les chapitres suivants, ce qui facilitera les comparaisons. Pour terminer, une analyse synthèse démontre les avantages et les limites des approches traditionnelles.

1.1 Origine des approches traditionnelles et du génie logiciel

À la fin des années 50, l'informatique est devenue de plus en plus populaire et s'est étendue à d'autres disciplines. Cet élargissement à un public non scientifique, a engendré la création du métier de programmeur. L'utilisateur exprimait ses besoins et le programmeur réalisait la solution informatique. Ce fût le premier écart qui sépara les professionnels de l'informatique et les utilisateurs. À la fin des années 60, les grands systèmes commerciaux ont démontré qu'il était difficile d'adapter à grande échelle, les principes jusque là adéquats. Les grands projets dépassaient les budgets et les échéanciers, ce fût alors « la crise du logiciel » [Ghezzi 1991].

On explora l'aspect de la gestion de projets, la composition des équipes, des outils et les standards de code. On en conclut que le logiciel est un produit complexe, qui doit être abordé comme les autres produits complexes tels que les ponts, les raffineries et les avions. La réduction du coût des machines et l'augmentation des coûts des logiciels ont permis de mettre l'emphase sur cet aspect économique [Ghezzi 1991]. Une étude sur la répartition des efforts démontra qu'au cours d'un développement logiciel, près de 50% des efforts sont investis dans la gestion et le support, 15% pour le design, 20% pour la réalisation et 15% pour les tests (voir Figure 1-1 : Répartition des efforts de développement [Tiré de NATO 1968]). La constatation du manque de structure pour ces projets a ouvert la porte à une nouvelle discipline nommée : génie logiciel.

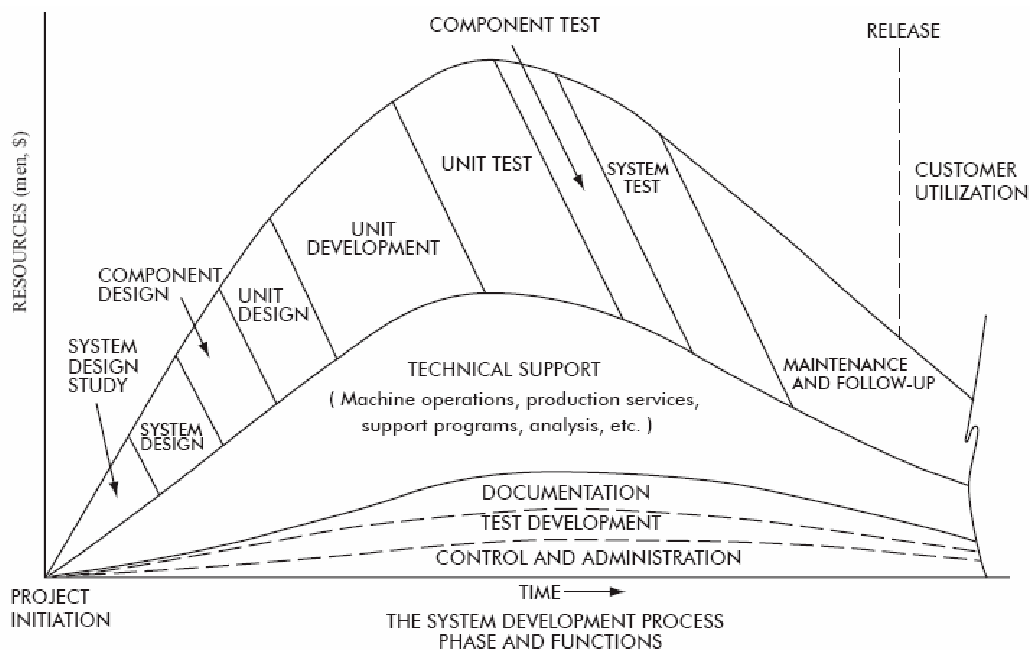


Figure 1-1 : Répartition des efforts de développement

[Tiré de NATO 1968]

1.1.1 Le domaine du génie logiciel

Le génie logiciel couvre un grand nombre de sujets, dont certains traitent spécifiquement de la gestion et du processus de développement. Le génie logiciel peut se définir comme l'application d'une approche systématique, disciplinée et quantifiable pour le développement, l'opération et la maintenance de logiciel. Il s'agit de l'application du génie au domaine logiciel. Plus spécifiquement, le cycle de vie du logiciel est l'ensemble des étapes rencontrées par le logiciel au cours de son existence. Il couvre son origine, sa création, sa maintenance et son retrait dans l'objectif d'appliquer de meilleures pratiques, tant individuelles qu'organisationnelles [SWEBOK 2004]. C'est principalement sur la création du produit que porte notre attention, afin de mettre en évidence les différences entre les approches traditionnelles et agiles.

1.2 Les processus de production dans une approche traditionnelle

La définition d'un processus sert à déterminer l'ordre des étapes et à connaître les critères de transition pour passer d'une étape à l'autre. Il faut être en mesure de connaître la séquence des choses à faire et jusqu'où elles doivent être faites. [Ghezzi 1991]. La définition du processus permet d'améliorer la qualité du produit et facilite sa compréhension.

1.2.1 La gestion de projets

Dans une approche traditionnelle, les projets informatiques abordent la gestion de projets comme les autres disciplines. On y retrouve la planification, l'organisation, le suivi, l'affectation du personnel, les contrôles et la direction. On cherche à livrer le produit attendu dans les temps et les budgets prévus [DGI 1990a].

La planification prévoit le déroulement, afin d'affecter les ressources requises et d'estimer les coûts et les efforts. Elle permet de présenter un échéancier et les budgets nécessaires, ce qui permet de conclure des ententes forfaitaires incluant la durée et un prix fixe.

L'organisation des équipes et du travail, varie en fonction de la méthode suivie. Chacun a son rôle et ses responsabilités. Lorsque la coordination est complexe, il est préférable d'établir un plan de communication, afin de s'assurer que l'information circule bien. L'organisation du travail structure les tâches du processus, suivant un modèle ou une méthode.

Le suivi est une tâche continuelle, qui analyse les données de contrôle pour informer de l'avancement aux différents intervenants concernés. Le suivi peut prendre la forme d'un tableau de bord comportant différents indicateurs pertinents.

L'affectation du personnel prévoit les besoins en ressources et procède aux démarches nécessaires pour réserver ou embaucher le personnel requis.

Les contrôles servent à fournir des indicateurs sur les réalisations. Les différents rapports d'avancement peuvent être représentés par des diagrammes ou des tableaux. Les vérifications sont basées sur la concordance avec la planification. Il est ainsi possible d'évaluer les écarts et réviser les cibles.

Diriger, consiste à mettre en œuvre les opérations et corriger les écarts afin de se rapprocher des cibles. La gestion des risques identifie et classifie les éléments pouvant affecter le projet. Il existe différents types de risques identifiables : politique, économique, social, technologique, éthique, légal et organisationnel. Au cours du projet, il faut suivre l'évolution des facteurs de risque et au besoin, appliquer les mécanismes d'atténuations.

1.2.2 Organisation du travail

Le processus de production présente la séquence d'exécution des tâches liées à la production du logiciel. Indépendamment du modèle, ces tâches sont similaires d'un processus à l'autre. Ce qui les distingue, porte sur le niveau de précision, la portée et l'ordre de réalisation.

Tous les processus de production logiciel passent par quatre grandes phases : l'initiation, le développement, l'implantation et l'exploitation. L'initiation vise à connaître la mission du système et à réaliser les travaux exploratoires qui vérifieront la pertinence du projet. Le développement est la phase qui prend en charge la réalisation du logiciel. Elle précise les besoins, réalise le produit et valide son fonctionnement. L'implantation rend le produit accessible à ses utilisateurs, par la mise en opération du logiciel dans son environnement de production. L'exploitation est la période de vie utile du produit au cours de laquelle des améliorations peuvent s'ajouter au produit.

Un ensemble d'éléments est nécessaire à un projet suivant une approche traditionnelle. Plusieurs livrables doivent être révisés et approuvés, afin de progresser dans les phases. Le Tableau 1-1 : Description des tâches du cycle de vie logiciel, présente les phases et les principales tâches qu'elles regroupent. La section description comporte des mots en caractère gras, qui représentent un exemple de livrables pour cette tâche. Ce tableau est un résumé inspiré de différentes sources [Alter 2001; Ghezzi 1991; DGI 1990b].

Étapes	Description (**Livrable en gras**)
Initiation – Quelqu'un désire le produit	
Étude de faisabilité	**L'étude de faisabilité** produit un document qui analyse globalement les besoins et définit le mandat et les limites du système. Elle explore diverses solutions possibles. Pour chacune d'elles, on évalue les ressources nécessaires, les coûts et les bénéfices.
Planification de projet	**Le plan de projet** se présente souvent sous forme de diagramme de Gantt et présente à travers le temps les différentes phases qui seront réalisées. Il précise les ressources utilisées, la durée et les coûts du projet.
Développement – On doit concevoir et créer le produit	
Analyse et spécifications des exigences	**Le document des spécifications requises,** analyse et précise les qualités internes requises pour juger de la conformité du produit. Le document doit être compréhensible, complet, clair et précis.
Spécification du design (architecture)	**Le document des spécifications du design,** correspond à la décomposition en modules et les relations qu'ils ont entre eux. Il précise aussi les principales données et certains algorithmes.
Analyse fonctionnelle	**Le dossier d'analyse fonctionnelle** détaille les unités de programmation à leur plus bas niveau, avant d'être codé. L'ensemble des éléments doit précisément y être spécifié. On y retrouve les maquettes d'écran, les données exploitées et le détail des algorithmes.
Programmation et tests unitaires	La programmation réalise **le code source,** qui est le livrable de cette étape. **Un plan de tests unitaires** doit aussi être produit pour valider le code.
Intégration et tests systèmes	L'intégration consiste à valider l'interaction des unités de programmation ensemble. **Un plan des tests systèmes** prévoit la vérification des échanges entre les unités et les modules du système. Les interfaces modulaires sont testées à ce niveau.
Documentation système	La documentation explique comment exploiter l'application. On retrouve généralement une **documentation technique** destinée à l'administration du système et une **documentation didactique** destinée aux utilisateurs. Cette dernière est utile lors de la formation et de la consultation.

Implantation – Le produit doit être mis en opération	
Plan d'implantation	L'implantation est l'étape de livraison du produit au client. Le logiciel doit être installé sur les machines de production réelle. **Le plan d'implantation** guide les étapes à suivre pour procéder à cette mise en opération.
Formation	Les utilisateurs doivent être formés sur le nouveau système, afin qu'ils connaissent les fonctionnalités disponibles dans leur nouveau système. **Le plan de formation** détaille la stratégie et les moyens utilisés. Il peut prendre la forme de **cours magistraux**, de coaching, de lectures, etc.
Conversion	Cette étape sert à alimenter le nouveau système à partir de données existantes. Celles-ci doivent être converties dans un nouveau format. **Un plan de conversion** doit être produit afin de spécifier les interfaces d'importation et l'exportation des données existantes.
Tests d'acceptation	Les tests d'acceptation sont réalisés par les pilotes ou les utilisateurs, afin de d'approuver le système. **Une liste des demandes de corrections** est rédigée dans le but de corriger les éléments non conformes aux attentes du client.
Vérification posthume	Lors de la fermeture du projet de développement, une vérification posthume révise le déroulement des différentes étapes. **Le rapport de fin de projet** explique les difficultés rencontrées et les bonnes pratiques à retenir. Cette révision permet d'améliorer ses connaissances en gestion de projets et en développement de systèmes.
Exploitation – Le produit doit être maintenu fonctionnel	
Suivi opérationnel et support	Le suivi opérationnel assure le fonctionnement du système. Des éléments de surveillance permettent de contrôler le fonctionnement. Le support consiste à répondre aux questions et à compléter la formation auprès de l'équipe en charge de l'exploitation ou des utilisateurs.
Maintenance	La maintenance du système permet de corriger les erreurs de conception et de compléter ou d'ajouter des fonctionnalités.
Décisions récurrentes : Continuer – Refaire – Terminer	Le système doit être évalué de manière récurrente pour confirmer sa pertinence. À travers le temps, de nouvelles opportunités peuvent comporter suffisamment d'avantages pour abandonner un système désuet.

Tableau 1-1 : Description des tâches du cycle de vie logiciel

1.2.3 Les modèles reconnus

À travers la littérature, il est possible de dégager plusieurs modèles de production logiciel aussi appelés « Cycle de vie logiciel ». Ces modèles servent de références théoriques [CTG 1998; Ghezzi 1991]. Ils mettent en relief les phases du processus et leur ordonnancement. Cette section présente divers modèles proposés.

Le modèle ad-hoc n'est pas un modèle documenté. Ce fût le premier moyen utilisé au début de l'aire informatique [Ghezzi 1991]. On y réfère pour signifier qu'il est possible de produire un logiciel sans suivre de modèle [CTG 1998]. Il s'est avéré inapproprié pour de gros logiciels et livrait des produits qui ne correspondaient pas aux attentes des utilisateurs [Ghezzi 1991].

Le modèle en cascade fût le standard industriel dans les années 70, il est apparu suite à la crise du logiciel [Ghezzi 1991]. Il présente une série d'étapes subséquentes réalisées l'une après l'autre [Royce 1970]. On apprécie que la documentation soit intégrée à chaque étape et que des vérifications soient faites sur tous les livrables. [Ghezzi 1991; CTG 1998]. Cependant, la rigidité du modèle est fortement critiquée. Il est difficile d'évaluer correctement le projet au départ, car peu d'informations sont disponibles. Les spécifications écrites permettent difficilement à l'utilisateur d'anticiper le système, ce qui limite les chances de rencontrer les attentes [Ghezzi 1991]. Concrètement, il est rare qu'un projet suive une série d'étapes de manière séquentielle, car cela a pour effet d'allonger les délais. Malgré tout, ce modèle reste à la base de plusieurs autres.

Le modèle en « V » est une première amélioration du modèle en cascade. Il a été créé pour palier le manque de réactivité du modèle en cascade. Il permet de créer des spécifications qui sont validées à un niveau de précision équivalente, ultérieurement dans le processus, formant ainsi un modèle en « V ». Il met en évidence la nécessité d'anticiper et de préparer l'étape suivante. Les éléments nécessaires à la validation sont préparés lors des spécifications. Ce modèle est devenu le standard des années 1980 [Wikipedia 2006f].

Le modèle itératif est une alternative pour livrer plus rapidement, requérir moins d'informations préalables et offrir plus de flexibilité. Le projet est divisé en itérations, chacune équivalant à des sous-projets ayant peu d'interactions. Chaque itération peut

présenter un produit plus rapidement. On obtient les impressions du client et on apporte les correctifs nécessaires. Au cours d'une itération, on suit un modèle en cascades [CTG 1998]. Ce modèle exige plus d'implication de la part du client. La communication et la coordination prennent un rôle central dans ce type de projet.

Le modèle par prototypage fût créé du fait qu'il est difficile d'obtenir les spécifications au début d'un projet. Ce modèle consiste à construire une version réduite ou théorique du système, afin d'obtenir les approbations. Les commentaires permettent de raffiner les spécifications [CTG 1998]. Ce modèle peut se greffer à un modèle en cascade ou itératif.

Le modèle en spirale a été conçu pour inclure les meilleurs éléments du prototypage et du modèle en cascade. Il introduit la notion d'évaluation des risques, pouvant mener à l'arrêt du projet si ceux-ci sont jugés majeurs [Boehm 1988]. Le terme « spirale » est utilisé pour décrire la forme que prend le modèle. Il passe par une série d'étapes similaires, au fur et à mesure que le système se précise. Il propose des solutions qui sont évaluées par le client puis enrichies dans un cycle ultérieur. Telle l'approche par prototypage, une première version est produite et raffinée suite aux évaluations. Contrairement au prototypage, le modèle est soigneusement conçu pour chaque version produite en suivant les étapes du modèle en cascade. Progressivement, en partant du centre de la spirale, une version plus complète est produite, ce qui représente un accroissement de la charge [CTG 1998].

Le modèle incrémental, parfois nommé évolutif, est un modèle itératif pour lequel chaque itération peut donner un produit fonctionnel qui peut être livré au client. Chaque itération comporte des modifications au design et de nouvelles fonctionnalités fortement couplées à celles existantes. La première livraison du produit est une version simpliste du système. On pourrait le comparer avec un noyau de systèmes qui évolue, les ajouts constituent ainsi un incrément. Il répond au besoin de flexibilité et de modularité, en livrant par sous-ensembles de fonctionnalités. Ce modèle intègre les dernières tendances en conception de produit. On part d'une base simple et on en fait émerger les spécifications et évoluer le produit. [Wikipedia 2006e]

Il existe d'autres modèles voués aux domaines scientifiques, répondant à des besoins particuliers. Ceux-ci ne seront pas abordés ici, car ils ne cadrent pas avec l'objectif de ce chapitre.

1.2.4 Adaptations des modèles

Il faut tenir compte des différentes variables pour choisir un modèle convenable et obtenir les bénéfices attendus. L'environnement organisationnel et la nature de l'application sont des facteurs importants pour choisir un modèle [CTG 1998; DGI 1990a].

La stabilité de l'environnement et des besoins, permet de choisir entre une approche prédictive ou adaptative. Un environnement est jugé stable lorsque les spécifications ne changent pas au cours de la vie du système. Il est ainsi possible d'établir dès le départ des spécifications précises. Les modèles prédictifs, comme la cascade ou la spirale sont adéquats.

D'autre part, un environnement turbulent se caractérise par des changements continuels à divers niveaux. Il est difficile de planifier les spécifications en raison des incertitudes. Ce type d'environnement nécessite une approche livrant rapidement, ce qui permet d'obtenir rapidement les impressions des utilisateurs et préciser leurs besoins. Les modèles incrémentaux avec prototypage et réutilisation, facilitent l'avancement des projets dans ce contexte [CTG 1998].

La nature de l'application est déterminante. La grosseur et le niveau critique du système sont d'autres facteurs importants dans le choix du modèle. Il exige plus de rigueur à tous les niveaux de conception, ce qui impose plusieurs vérifications. À l'opposé, des petits projets pilotes pour vérifier une preuve de concept ou explorer une opportunité incertaine, correspondent mieux au modèle adaptatif.

1.2.5 Méthodes populaires

Une méthode est l'application concrète d'un modèle et détaille la démarche à suivre. Au cours des années 1980, les méthodes Merise, SADT et Productivité PLUS de DMR furent fortement répandues. Ces approches séparaient les traitements et les données, ce qui

convenait aux technologies en vigueur au cours de cette période. Au cours des années 1990, les langages orientés objets, modélisés par les diagrammes UML, gagnèrent en popularité. Le gabarit de méthode RUP propose un modèle itératif basé sur de courtes livraisons [IBM 2006].

1.2.6 Validation du processus

Afin de s'assurer que le processus d'organisation correspond aux bonnes pratiques en vigueur dans l'industrie, il est possible de le certifier selon une norme. Une norme est un document, établi par consensus et approuvé par un organisme reconnu, qui fournit pour des usages communs et répétés, des règles, des lignes directrices ou des caractéristiques, pour des activités ou leurs résultats, garantissant un niveau d'ordre optimal dans un contexte donné [OQLF 2006a].

Il existe plusieurs normes mais, dans le domaine logiciel le modèle CMM et la norme ISO-9000 se sont démarqués. Le CMM, plus précisément le SW-CMM du CMMi pour le développement logiciel, propose un modèle composé de 5 niveaux de maturité (voir l'annexe D). L'organisation améliore son processus jusqu'à l'atteinte du niveau optimal [SEI 2005]. Pour sa part, la norme ISO-9000 vise principalement la satisfaction de la clientèle [Wikipedia 2006a].

La certification est une référence universelle, qui comporte principalement des avantages commerciaux pour qualifier les fournisseurs. Les efforts de certifications sont importants et doivent comporter une valeur pour l'entreprise. Autrement, une simple amélioration des processus peut être moins coûteuse et tout aussi efficace.

1.3 Le produit dans une approche traditionnelle

Dans les approches traditionnelles, le produit est le terme désignant à la fois le logiciel et sa documentation. On doit s'assurer que le logiciel correspond aux attentes et qu'il fournit un produit complet, pouvant être transmis à un tiers, qui s'occupera de son entretien.

1.3.1 Spécifications du produit

Les spécifications servent à décrire le système en fonction des besoins du client. Elles listent les fonctionnalités du système. Au cours du développement, il est possible de valider la conformité du logiciel en fonction de ces spécifications.

Une partie des spécifications couvrent le comportement opérationnel et algorithmique. Elles conceptualisent l'utilisation et les réactions du logiciel pour expliquer son fonctionnement. Ces diagrammes servent d'outils de communication entre les informaticiens et les pilotes.

L'autre type de spécification décrit la conception du logiciel. Celle-ci touche l'aspect architectural de l'application, tant sur le plan des données, des fonctionnalités que sur la répartition du déploiement des modules. Ces diagrammes s'adressent plus particulièrement aux informaticiens.

1.3.2 La qualité du produit

La qualité du produit peut être définie par ses attributs externes et internes. La différence s'établit par ce qui peut être perceptible de l'extérieur et ce qui est caché. On remarque souvent l'interface utilisateur pour juger de la convivialité du système. Une bonne documentation écrite ou une aide en ligne complète, sont des éléments perceptibles par les utilisateurs. Il est aussi possible d'inclure les notions de performance et de sécurité pour juger la qualité d'un logiciel.

L'autre facette de la qualité touche les caractéristiques internes du produit, c'est-à-dire la façon dont le logiciel est construit. La qualité de la rédaction du code source, tel qu'un bon découpage modulaire et de bons commentaires explicatifs, aura un impact bénéfique sur son entretien et ses améliorations.

Il est possible de s'appuyer sur certaines mesures pour fixer les limites d'acceptation du produit. Par exemple, le temps de réponse maximal pour obtenir des informations peut être déterminé. D'autre part, un ratio de lignes de commentaires peut aussi être une mesure retenue. Certaines organisations ont des groupes d'assurance qualité et de gestion de la

configuration qui se préoccupent essentiellement de ces aspects, pour accroître la qualité du produit.

1.3.3 La standardisation

Au niveau logiciel, la standardisation cherche à simplifier la conception afin de faire des gains de productivité et simplifier les échanges. On peut dire qu'un produit suit des standards lorsque la conception et la réalisation ont été effectuées à partir de spécifications techniques reconnues dans le but d'uniformiser [OQLF 2006b]. Cette uniformisation peut venir de l'industrie ou être établie par l'organisation.

Les standards se retrouvent à différents niveaux. Il est possible de suivre des standards au niveau de la nomenclature, des règles de codage, des interfaces utilisateurs ou des raccourcis claviers. Au niveau technologique, on en retrouve au niveau des protocoles de communication et des langages. Plus récemment, UML et les patrons de conception (patterns) sont aussi devenus des standards au niveau de la conception. Suivre un standard évite les pertes d'efforts à reproduire quelque chose d'existant. Ils proposent des solutions convenables, assurant une uniformité avec d'autres systèmes ou interlocuteurs.

1.4 Les développeurs dans une approche traditionnelle

Les développeurs sont les principaux acteurs des projets de développement. On associe souvent un titre à une fonction. Cette section présente les différents rôles et les types d'outils utilisés.

1.4.1 Les rôles et responsabilités

Les approches traditionnelles spécialisent les tâches. Ces tâches sont réalisées par des individus ayant un rôle précis. Il existe plusieurs titres tels que : architecte, analyste, programmeur, etc. En général, le titre est associé au niveau de formation ou de spécialisation de cette personne. La profession ne protège aucun titre, ce qui apporte beaucoup de confusion à ce sujet. Le tableau suivant présente un résumé des titres usuels et les affectations attribuées [DGI 1990a].

Titre	Affectation
Équipe informatique	
Chef de projet	Il est en charge de mettre en œuvre les différentes tâches et est le point de contact pour les communications. Il est souvent l'entremetteur entre le client et l'équipe de développement.
Analyste fonctionnel ou Architecte	Cette personne produit l'architecture de haut niveau. Il coordonne aussi les réalisations techniques.
Analyste organique	Avec l'aide du pilote ou d'utilisateurs, cet individu est en charge de détailler les modules.
Analyste - Assurance qualité	Son travail consiste à aider et proposer des règles pour améliorer la production, afin d'en assurer la conformité.
Programmeur	Cette personne réalise le code source à partir des spécifications demandées.
Testeur	Cette personne réalise les jeux d'essais prévus, pour valider le comportement des modules programmés.
Opérateur	Cet individu assure les opérations du système. Il assure la veille du système, afin de contrôler les opérations au besoin.
Rédacteur	Il est attitré à la rédaction de la documentation système. Cette fonction peut être combinée au support à la clientèle.
Équipe non informatique	
Client	Terme générique qui réfère à la fois au décideur, au pilote et à l'utilisateur.
Directeur	A un rôle de décideur, il gère principalement les budgets.
Pilote - Expert du domaine	Il détient l'expertise du domaine. Il est en charge du transfert de connaissance pour la spécification du produit. C'est aussi le principal approbateur des éléments livrés.
Utilisateur	Personne qui exploite le produit.

Tableau 1-2 : Principaux titres et rôles des approches traditionnelles

De plus, à ces nombreux rôles, s'ajoutent différents comités qui ont principalement des tâches décisionnelles. Ils veillent à la planification et aux approbations. Ces groupes sont composés de gens de différentes spécialités. Plus il y a d'individus impliqués, plus les efforts de communication sont importants.

1.4.2 Les outils logiciels

Les équipes de développement utilisent plusieurs outils logiciels dans la réalisation de leurs tâches. Le livrable détermine souvent l'outil qui sera nécessaire. Par exemple, lorsqu'il s'agit de rédiger un rapport, un logiciel de traitement de texte est suffisant.

Dès le début, il est possible d'utiliser un logiciel de planification de projet pour produire un WBS ou un diagramme de Gantt pour prévoir les besoins en ressources. Au cours de la réalisation, un logiciel de suivi de projet peut aider à faire les rapports d'avancement.

Au niveau informatique, la conception et le design du système peuvent être pris en charge par un outil de modélisation ou de conception assistée (CASE) intégrant des générateurs de code pour éliminer l'étape de codage. L'éditeur de code et le compilateur demeurent au centre du développement. Ils construisent et déboguent les unités de programmation. Les versions récentes de ce type de logiciel proposent des interfaces graphiques très complètes et intègrent des liens avec des logiciels de contrôle de sources, des diagrammes, la génération de la documentation système et l'automatisation des tests.

En support à la production, les outils de tests aident au suivi des correctifs répertoriés au cours des essais. Les analyseurs de code source, aident à extraire différentes mesures pour valider sa conformité vis-à-vis les standards et règles établies. Lors de la livraison du produit, des outils de mise en production ou d'aide à la composition d'ensembles livrables (packages) peuvent aussi être utilisés.

Au niveau de la coordination des équipes de travail, plusieurs échanges se font de manière officielle. Dans ce cas, le logiciel de courrier électronique ou les partagiciels sont intégrés au plan de communication.

Plusieurs types d'outils sont disponibles. Pour chaque type d'outils, il existe différents produits de fournisseurs concurrents. Le choix du produit est délicat, car les connaissances acquises peuvent devenir obsolètes si le produit est abandonné par le fournisseur. Les développeurs doivent constamment mettre à jour leurs connaissances, créant ainsi leur expertise. Lorsque qu'un développeur se dédie à un produit ou une technologie, on le nomme spécialiste. Les efforts d'apprentissage et d'entretien des connaissances sont importants. Un développeur ne peut être spécialiste que dans très peu de domaines à la fois. À l'opposé, un développeur peut s'informer et suivre plusieurs sujets, ce qui en fera un généraliste. La différence est au niveau de la répartition des efforts d'apprentissage.

1.5 Le client dans une approche traditionnelle

Pour l'équipe informatique, le client est un terme générique pour désigner à la fois la direction, les gestionnaires, les pilotes et les utilisateurs de l'organisation qui les emploie. Le rôle principal du client est de communiquer ses besoins pour guider l'équipe de développement.

La direction et les gestionnaires sont responsables du financement et du suivi des opérations. Ils déterminent les budgets et font le suivi de coûts pendant l'avancement des travaux. Ils s'occupent principalement des relations d'affaires entre les deux organisations. Lorsque le développement est fait par une firme externe, un processus normal de soumission est utilisé. Le client rédige un cahier de charge et va en appel d'offre. Les différents fournisseurs déposent leur soumission en fonction de leurs estimations. Le client fait son choix et conclut une entente contractuelle pouvant varier entre une facturation horaire et une entente forfaitaire.

Le pilote est le principal responsable du transfert de connaissances. Il doit être très bien informé des opérations que le système supportera. Il est possible que plusieurs pilotes soient nécessaires pour couvrir toutes les fonctionnalités d'un système. Le pilote est généralement un utilisateur représentatif et expérimenté. Il définit les besoins et valide la conformité du fonctionnement. Les corrections à apporter peuvent mener à des demandes de changements, qui nécessitent parfois la renégociation du contrat. Dépendamment des approches, cette tâche peut être exigeante pour le pilote. Dans une approche en cascade, on doit prévoir des efforts principalement au début et à la fin du développement. Dans une approche itérative, chaque itération requiert le pilote, ce qui en fait une tâche très soutenue.

Les utilisateurs sont les bénéficiaires du système. Comme leur nom l'indique, se sont eux qui utiliseront couramment le système. C'est le principal groupe de personnes qui jugera de la convivialité du système. Si possible, il est préférable d'implanter graduellement le système par secteur. Ce type d'approche permet de corriger certaines lacunes, avant que le système ne soit fortement sollicité.

1.6 Analyse des approches traditionnelles

Les approches traditionnelles comportent des avantages et des limitations. Cette section propose une synthèse des éléments favorables à leur utilisation et les limites pour lesquelles un projet n'aurait pas avantage à suivre ces approches.

1.6.1 Application favorable des approches traditionnelles

Les approches traditionnelles sont structurées et prédictives. Lorsqu'un projet exige ces éléments, les approchent traditionnelles sont recommandées.

La gestion de projets fait le suivi des principales variables : temps, argent et produit. Elle intègre la gestion des risques et un plan de contingence. Le processus rigoureux comporte suffisamment de vérification pour fournir un produit de qualité. Il est adéquat pour les projets critiques ou d'envergure.

L'organisation du travail convient aux organisations hiérarchisées, qui commandent et contrôlent l'exécution des travaux. Elle fournit un bon suivi et permet la reddition de compte à la haute direction. La standardisation et l'uniformisation des pratiques permettent de les optimiser. Le processus établi peut être certifié.

Les documents écrits permettent la coordination de grosses équipes multidisciplinaires et géographiquement distribuées. La spécialisation des tâches permet l'accès à des experts. La planification permet de garantir la disponibilité des ressources lorsque nécessaire. L'impartition avec un pays étranger est aussi possible.

Ces avantages rassurent et anticipent les efforts requis, permettant des ententes forfaitaires. Ce type d'approche est aussi conseillé lorsqu'une organisation a une vision à long terme du développement de ses systèmes. Le retour sur l'investissement se calcule sur une longue période en fonction des sommes impliquées.

1.6.2 Limitation des approches traditionnelles

Les approches traditionnelles ont aussi leurs limites. Certains facteurs peuvent nuire à l'efficacité des ces approches.

Les environnements changeants demandent des adaptations constantes. Avec une planification rigoureuse, il est difficile d'apporter des changements à cause de leurs impacts. Plus une modification est faite tardivement, plus elle est coûteuse. Elle requiert de modifier la chaîne des étapes préalables. Seules les modifications les plus importantes sont effectuées.

Certains projets peuvent requérir moins d'effort de gestion. Les suivis et rapports alourdissent la charge administrative. Les communications écrites sont coûteuses et longues à produire. Il faut tenir compte de l'ampleur et de la complexité du projet. Si le produit est livré dans un trop grand délai, il est possible que la situation du client ait évolué et que l'on doive modifier le système.

L'expérience de l'équipe peut faire diminuer les risques et contribuer à minimiser les exigences de suivi. Les approbations et le manque d'autonomie des équipes peuvent ralentir le projet. Une organisation peu hiérarchisée, ayant une culture collégiale, intégrera difficilement une structure traditionnelle.

1.7 Conclusion

Ce chapitre rappelle les caractéristiques des approches traditionnelles. Le développement est un des aspects du génie logiciel qui vise à structurer ses pratiques. Le logiciel étant un produit complexe, les approches proposées s'inspirent de d'autres domaines d'ingénierie.

La gestion de projets prévoit le déroulement des opérations et les suit pour qu'elles correspondent au plan. L'organisation des tâches peut adopter différents modèles. Lorsque le processus d'une organisation est suffisamment défini, il peut être certifié selon qu'on cherche à optimiser les opérations ou à satisfaire sa clientèle [Cockburn 2004].

Le produit résultant doit correspondre aux spécifications du client et le processus valide la conception de chaque élément. En plus des fonctionnalités incluses, la qualité du logiciel peut être définie selon plusieurs mesures. Il est possible de réduire les efforts et d'améliorer la qualité du produit en suivant des standards.

L’équipe de développement est, elle aussi, structurée. Chaque intervenant occupe une fonction dans la chaîne de production. Cette séparation permet d’affecter les tâches à des spécialistes. Chaque rôle correspond à des outils que le développeur doit maîtriser, ce qui requiert aussi beaucoup de connaissances.

Pour sa part, le client reste le décideur. Il initie le projet et peut y mettre un terme en dépit de ses ententes contractuelles. Il communique ses attentes à l’équipe de développement. Dépendamment de la méthode, il peut être plus ou moins impliqué, le plus important demeure sa satisfaction.

Ce chapitre dresse un bilan rapide des approches traditionnelles. Les sujets traités sont groupés de façon similaire au chapitre sur les approches agiles, afin de faciliter leur comparaison. De cette manière, les similitudes et les divergences de visions sont plus facilement perceptibles.

Chapitre 2: Caractéristiques des méthodes agiles

Ce chapitre présente les éléments de base qui définissent l'agilité. Il établit quelle est l'offre de ces approches dans le domaine du génie logiciel. Il positionne ces approches par rapport aux modèles établis, ainsi que leurs limitations. Il présente les différentes sources de supports agiles. Finalement, il dresse le bilan des points communs et des différences avec les approches traditionnelles.

2.1 Les bases de l'agilité

2.1.1 Définition du terme « Agile »

Présentement, l'attrait pour le terme « Agile » est répandu. Chaque secteur d'activité peut attacher ce terme à un paradigme. Il doit être interprété selon le domaine auquel il est appliqué. Ce terme n'est pas breveté, il n'y a pas de limitation sur son utilisation, ce qui peut donner lieu à différentes définitions apparentées.

La définition littéraire du mot « Agile » ne suffit pas. Il faut considérer ce terme comme étant la définition la plus proche, pouvant définir la capacité de répondre aux changements que proposent ces approches. L'agilité exige de s'adapter au changement inhérent aux environnements turbulents. Ces changements pouvant être au niveau des spécifications, des intervenants, des procédures et surtout, dans le domaine logiciel, des technologies. Appliqué au génie logiciel, Jim Highsmith définit l'agilité ainsi : « L'agilité est l'habilité conjointe de créer et de répondre aux changements d'un environnement turbulent au profit de l'entreprise. » [Traduit et tiré de Highsmith 2002a]

2.1.2 Origines des approches agiles

Avec l'évolution des technologies, les méthodes traditionnelles de développement logiciel s'avèrent trop lourdes dans plusieurs situations. Une opportunité pour trouver des alternatives structurées était souhaitable. Au cours des années 1990, différents praticiens ont mis sur pied des méthodes qui changeaient la conception traditionnelle de mener un projet de développement logiciel.

Ces premiers « Agilistes[1] » se sont rencontrés au Centre de ski Snowbird en Utha, le 13 novembre 2001. Cette rencontre avait pour objectif d'échanger et de trouver les points communs des méthodes promues par les dix-sept participants présents. C'est lors de cette rencontre qu'ils ont rédigé le manifeste agile [AA 2005b], pierre d'assise du mouvement. Par la suite, ils ont regroupé une liste de principes pour enrichir le manifeste. L'ensemble se veut un cri de ralliement, afin de faire connaître leur point de vue à l'industrie du logiciel [Fowler 2002].

L'origine du terme « Agile » est issue de l'ingénierie et des milieux manufacturiers qui ont introduit la notion de gestion agile. Lorsqu'il est venu le temps de définir le mouvement de pensée rassemblant les différentes méthodes, ce terme fût adopté. Il transpose le transfert d'expertise provenant des autres domaines d'ingénierie. Du coup, les termes « léger » ou « maigre » pouvant être perçus comme étant simplistes et négligés, furent écartés.

Les différentes approches agiles[2] sont principalement définies par des méthodes[3] qui mettent en œuvre différentes pratiques[4]. Unitairement, les pratiques ne sont pas nouvelles, il s'agit pour la plupart d'un retour vers la simplicité. L'agilité amène une réflexion sur l'objectif du travail et l'esprit dans lequel il doit être réalisé.

[1] Agiliste : Individu adhérant aux valeurs et aux principes agiles
[2] Approche agile : Application de la philosophie des principes agiles.
[3] Méthodes agiles : Ensemble de pratiques agiles structurées formant une méthode.
[4] Pratique agile : Action concrète qui actualise les valeurs agiles, telle que la programmation en pair.

2.1.3 Le manifeste agile

Les méthodes agiles se définissent par leur adhésion aux critères du manifeste qui constitue la pierre d'assise des approches agiles. Les projets de développement sont réalisés avec des ressources limitées qui imposent des choix. Le manifeste souligne les préférences des Agilistes. Il communique les valeurs des fondateurs et leurs préoccupations, qui les menèrent à ces conclusions: Une méthode propriétaire d'entreprise peut être agile, si celle-ci respecte le manifeste et les principes agiles. Le manifeste est composé de quatre énoncés. Il débute et se termine par deux autres phrases tout aussi intéressantes.

Nous sommes à découvrir de meilleures manières pour développer des logiciels en les pratiquant et en aidant les autres à le faire. À travers notre travail nous en sommes venus à valoriser : **les individus et les interactions,** davantage que les processus et les outils, **les logiciels fonctionnels,** davantage que la documentation compréhensive, **la collaboration avec le client,** davantage que la négociation de contrat, **la réponse au changement,** davantage que le suivi d'un plan. Bien que l'élément de droite soit important, l'élément de gauche l'est davantage

Tableau 2-3 : Le manifeste agile.

[Traduit et tiré de AA 2005b]

La phrase d'introduction comporte beaucoup d'éléments. Elle met en évidence que les méthodes agiles sont encore à l'étape de la découverte. Que leurs supporteurs sont des praticiens et qu'ils ont un esprit de collaboration, afin d'aider les autres dans leur apprentissage. Ils veulent ainsi introduire cette approche comme une alternative prometteuse plutôt qu'une solution ultime qui doit être inculquée. Le manifeste se poursuit avec quatre énoncés qui mettent l'accent sur différents aspects des projets de développement logiciel.

Les individus et les interactions, davantage que les processus et les outils. La cohésion d'une équipe est plus importante que le suivi des conventions. Cette phrase s’attarde à la qualité des individus impliqués et l’esprit de collaboration qui en fait un véritable travail d’équipe. Cet élément reflète la synergie et la motivation qui doit laisser au second plan la rigidité du processus ou la complexité des outils. On laisse de côté l'élitisme, souvent lié aux connaissances techniques, au profit des qualités interpersonnelles et à la motivation.

Les logiciels fonctionnels, davantage que la documentation compréhensive. Un logiciel partiellement fonctionnel, est toujours plus valable qu’une documentation soignée. L’objectif d’un développement logiciel est de produire un logiciel, la documentation doit être considérée comme un support complémentaire aidant à sa compréhension. Il est préférable d'investir ses efforts sur le produit, plutôt que sur les éléments qui le décrivent.

La collaboration avec le client, davantage que la négociation de contrat. Le client et le fournisseur sont des partenaires qui partagent un risque. L’agilité vise la collaboration des deux parties pour atteindre un objectif commun. Le client doit s’impliquer et diriger l’équipe dans les étapes de réalisation. Il doit s'approprier l'évolution de son produit. L’agilité évite le gaspillage des négociations et des demandes de changement. Elle propose d'investir ces efforts de manière constructive.

La réponse au changement, davantage que le suivi d'un plan. L’agilité reconnaît l’apprentissage fait au cours d’un projet. Que tout n’est pas connu d’avance et que des adaptations sont inévitables. Le changement est intégré au modèle de développement. La planification originale ne doit pas être trop détaillée, car certains aspects ne seront probablement jamais réalisés. Ceci évite de détailler inutilement des éléments trop tôt dans le projet.

Avec la dernière phrase, le manifeste conclut en établissant des priorités. Les Agilistes reconnaissent l’importance des éléments de droite, mais jugent que les éléments de gauche doivent être favorisés. Ce n’est pas un désaveu ou un abandon de la pratique, les Agilistes prônent la transparence et avouent leur préférence. Dépendamment des méthodes, ces préférences sont aussi nuancées.

2.1.4 Les principes agiles

Le manifeste ne suffit pas à communiquer toute la richesse de la réflexion des Agilistes. Une série de principes précisant leur vision, suit la rédaction du manifeste [AM 2006].

Nous suivons ces principes:

- **Notre plus haute priorité est de satisfaire le client en lui livrant rapidement et de façon continue, un logiciel de qualité.** Mettre le logiciel rapidement à l'épreuve pour connaître les ajustements décelés par le client. Il peut ainsi apporter son retour d'expérience.

- **Accepter les changements de besoins, même lors du développement.** Toutes les méthodes agiles incorporent la gestion des changements. Les modifications sont intégrées lors de la planification d'une itération.

- **Les processus agiles exploitent les changements pour augmenter les avantages compétitifs du client.** En cours de projet, les opportunités avantageuses pour le client peuvent être incluses à son avantage.

- **Livrer fréquemment un logiciel fonctionnel, en visant les délais les plus courts, de quelques semaines à quelques mois.** De cette manière, nous obtenons une preuve d'avancement du produit et minimisons les risques. Ces livraisons rapides motivent l'équipe, ce qui aide à maintenir un bon rythme de travail.

- **Gestionnaires et développeurs doivent travailler ensemble, de façon quotidienne, pour toute la durée du projet.** Peu de contrôle formel existe, la reddition de compte n'est pas une activité prioritaire. Des rencontres quotidiennes augmentent la qualité de la communication et permettent le suivi au gestionnaire.

- **Bâtir des projets autour d'individus motivés.** Le cœur des projets agiles passe par les individus impliqués. Les gens doivent avoir l'énergie et le courage nécessaire que demandent de tels projets.

- **Donner leur l'environnement et le support nécessaire et ayez confiance qu'ils feront le travail.** Des équipes autonomes et autorisées à prendre des décisions techniques peuvent optimiser les résultats. Les gestionnaires doivent accepter de faire confiance et déléguer.

- **La méthode la plus efficace pour transmettre l'information à l'équipe de développement et à l'intérieur de celle-ci, est par conversation de personne à personne.** L'interaction directe entre les individus est le médium de transmission le plus simple et efficace. La communication non-verbale peut être perçue et des précisions apportées.

- **Un logiciel fonctionnel est la mesure principale de l'avancement.** Mettre à l'essai le produit est la seule preuve valable de progression. Le client doit constater l'avancement des travaux. Les schémas ne sont que des hypothèses tant que le système n'est pas réalisé et testé.

- **Les processus agiles favorisent le développement maintenable.** L'ensemble du processus suit différentes règles qui simplifient l'entretien. Ces pratiques augmentent la rapidité d'intégration des changements.

- **Les responsables, les développeurs et les usagers devraient pouvoir conserver un rythme constant indéfiniment.** Pour assurer une progression constante, la charge de travail doit être soutenable. Il faut prendre en considération les besoins humains qui exigent des temps d'arrêt. Les procédures de fin d'itérations comblent ce besoin.

- **Une attention continuelle à l'excellence technique et un bon design, augmentent l'Agilité.** Une bonne conception facilite les modifications. La charge mentale est moins imposante. Il est plus facile de comprendre le code afin de le modifier. Cette attention a un impact direct sur les coûts d'entretien.

- **La simplicité ou « l'art de minimiser la quantité de travail fait inutilement » est essentielle.** La simplicité et la conception minimale évitent des travaux inutiles. Les travaux en lien avec des hypothèses ou des plans à long terme, doivent être évités. Le code sera adapté lorsque nécessaire dans un contexte concret.

- **Les meilleures architectures, exigences et designs surgissent d'équipes auto-organisées.** L'équipe doit pouvoir prendre les décisions techniques, car elle est la mieux placée pour prendre ce type de décision. Leurs compétences technologiques doivent leur permettre l'autonomie requise pour optimiser les résultats dans ce domaine de compétences. Les développeurs sont souvent fiers de leur travail et recherchent l'excellence technique.

- **À intervalles réguliers, l'équipe réfléchit sur une façon de devenir plus efficace, puis elle adapte et ajuste son comportement en conséquence.** Combiné avec les temps d'arrêt entre les itérations et sa capacité d'autogestion, l'équipe est appelée à corriger ses pratiques dans l'objectif de trouver un équilibre convenable (sweet-spot). Il n'y a pas de méthode ou de culture uniforme, celle-ci doit trouver son équilibre au cours de différentes itérations, afin que tous soient satisfaits des conventions établies.

- **Il existe plusieurs méthodes connues, cadrant dans ces valeurs et principes.** D'autres peuvent aussi avoir un succès moins rayonnant et parfaitement correspondre à ces critères qui en font une méthode agile.

Ce qui définit l’agilité est l'adhésion à ces principes. Appliquer cette philosophie est plus importante que le suivi de standard. Ce principe rejoint l’énoncé du manifeste qui indique qu’ils aident les gens dans cette découverte d’efficacité.

Ensemble, le manifeste et les principes définissent les valeurs agiles selon l’alliance. Ils constituent les bases pour les méthodes désirant porter l’appellation. Dépendamment des pratiques implantées, le niveau d’attente de ces valeurs peut varier. La prochaine section propose une analyse approfondie des bases de l’agilité.

2.2 L’offre agile

L’esprit entourant les approches agiles est souvent comparé aux sports d’équipes. L’une des approches s’appuie littéralement sur le soccer [Schwaber 1996]. D’autres proposent une analogie avec l’escalade en tant que jeux coopératifs, car elle comporte plusieurs similitudes. La coopération des autres est nécessaire pour atteindre un but. On ne connaît pas la voie précise à suivre, car elle est découverte au cours de la progression. Nous progressons rapidement avec des pistes sûres et lentement dans les pistes incertaines [Cockburn 2001].

Afin de définir l'offre des approches agiles, nous allons premièrement préciser certaines perceptions. Ensuite, on présente une analyse sur l’apport qu’elles offrent au domaine de l'industrie du logiciel selon différentes visions. Finalement, nous situerons l’agilité dans l’offre globale des différentes approches en génie logiciel.

2.2.1 Perceptions initiales

Lors d’un premier contact avec le manifeste et les principes agiles, la perception des gens diffère en fonction de leurs expériences. Généralement, les gens adhèrent aux valeurs agiles mais, anticipent certaines lacunes ou demeurent perplexes. Avant de pousser l'analyse, je désire préciser certaines perceptions erronées.

On pourrait croire que les méthodes agiles sont des « anti-méthodes » négligées, sans structure de processus. Qu'elles constituent l’antithèse des méthodes rigoureuses. Que leurs

adeptes sont des Hackers ou des Cow-boys. Que l'aspect itératif consiste à coder et corriger.

En fait, l'alliance agile n'a pas de droit sur l'appellation « Agile » et ne peut limiter son utilisation. N'importe qui peut utiliser ce terme, ce qui porte à confusion. Les méthodes associées à l'alliance agile comportent des modèles de processus et des pratiques à suivre. Leurs pratiques sont exigeantes et demandent beaucoup de discipline, ce qui est loin d'être à négliger [Fowler 2001].

On pourrait croire que l'objectif principal des approches agiles est la rapidité de livraison. Que le code est programmé plus rapidement par des techniques spéciales ou des outils de productivité.

En fait, les gains de temps se font par des livraisons réduites respectant des échéances fixes. Ces méthodes cherchent à minimiser la lourdeur des échanges d'information et les tâches inutiles. Les équipes sont plus autonomes, ce qui diminue les délais d'approbation. Tous ces éléments contribuent indirectement à livrer plus rapidement.

De plus, certains pourraient croire que le produit est de meilleure qualité et contient moins de défauts. Que le client est plus satisfait et que le projet est moins risqué.

En réalité, la qualité du produit n'est pas nécessairement meilleure avec ces approches. Une bonne équipe de développeurs évoluant dans un environnement traditionnel, réalisera un produit d'aussi bonne qualité. Les mêmes bonnes pratiques peuvent être utilisées dans les deux types d'approches. L'intégration des tests dans le développement contribue à améliorer la qualité du produit. L'intégration des changements répondant mieux aux besoins, satisfait davantage le client.

2.2.2 L'apport des approches agiles

Cette section vise à mettre en relief l'apport des approches agiles selon différentes visions. Il est intéressant de noter que ces approches touchent aussi le domaine de la psychologie et de la sociologie. Les projets agiles se réalisent dans un état d'esprit qui tient compte du facteur humain. À travers différents ouvrages de synthèse, plusieurs auteurs ont fait

ressortir des éléments communs [Ambler 2004; Highsmith 2002a; Augustine 2003]. En suivant l'ordre des énoncés du manifeste, nous faisons ressortir l'apport pour les développeurs, le produit, le client et la gestion de projet.

2.2.2.1 L'apport agile pour les développeurs

La première phrase du manifeste touche directement les individus et les interactions. Au-delà des outils, il y a les gens. Les développeurs impliqués dans un projet agile doivent être prêt à vivre une expérience de groupe. La cohésion de l'équipe est au centre du potentiel de réussite du projet.

En dehors des connaissances techniques et des règles méthodologiques, l'aspect humain est considéré [Cockburn 2000]. Différents éléments de ces méthodes s'attardent aux besoins de nature psychologique et sociologique. Différentes pratiques recherchent la sécurité, la reconnaissance des pairs et l'accomplissement personnel par la créativité. [Maslow 1943]. Cette préoccupation s'assure d'obtenir la meilleure contribution[5] de chacun, tout en conservant la motivation. La planification limitée sur de courtes échéances conserve le focus et permet d'être capable de concrétiser l'apport du travail journalier.

Les méthodes agiles s'appuient sur les individus. Elles ont besoin de gens compétents et motivés. Elles s'attardent aux attitudes plutôt qu'aux aptitudes [Arsenault 1997]. On préfèrera un développeur moyen et motivé plutôt qu'un expert individualiste. En diminuant les efforts de communication, on favorise l'accès aux informations [Cockburn 2001]. On contribue au partage des connaissances techniques des experts et à la compréhension générale du système. Les connaissances de l'individu s'améliorent continuellement, ce qui est une source de motivation.

L'intégration de la conception, de la réalisation et des tests, élimine des transferts de connaissance lourds et coûteux. Elle responsabilise davantage les personnes en charge, ne pouvant ainsi invoquer des carences dans les spécifications. On évite ainsi la déresponsabilisation liée au transfert (throw over the wall). Les développeurs intègrent

[5] Selon [Arsenault 1997], la performance humaine au travail est liée à 20% à son savoir-faire et 80% à son savoir agir

différents champs de compétences. En diminuant le cloisonnement des tâches, on facilite la relève lors d'absence ou de départ.

Les développeurs procèdent à l'apprentissage du domaine d'affaire du client. Au fils des rencontres, le client amène son retour d'expérience avec le produit. Les deux parties se connaissent mieux et diminuent leurs préjugés. Le client est plus sensible aux contraintes techniques. Cette collaboration permet d'optimiser les solutions proposées. Il en résulte un réel partenariat.

L'utilisation de délimiteurs fixes, améliorent le moral de l'équipe. En utilisant des périodes de temps fixe, les gens ne pourront être découragés par une limite continuellement repoussée. Cette caractéristique a l'avantage de banaliser l'horaire, pour en simplifier la gestion et favoriser la présence des participants. De plus, elle force la prise de décision pour prioriser les tâches.

2.2.2.2 L'apport agile pour le produit

La seconde phrase du manifeste insiste sur un logiciel fonctionnel. Les fonctionnalités couvertes par le développement du produit évoluent et s'adaptent aux changements. C'est principalement ce qui démarque les produits issus des approches agiles.

Souvent implanté dans les technologies orientées objets, l'utilisation de patrons de conception (design patterns) est encouragée, car ils permettent de simplifier les communications. Les autres techniques d'amélioration de la productivité, telles que la réutilisation ou l'approche par composants sont appliquées selon les méthodes. Les métriques de codes établies sont aussi applicables. Leur application est rarement discutée, car cette facette du développement est souvent laissée à la discrétion de l'équipe de développement.

Les qualités internes du produit sont importantes. En adoptant un design qui encourage l'adaptabilité, l'ajout de fonctionnalités est plus facilement intégrable. L'assurance qualité et les tests effectués à tous les cycles, minimisent les défauts. Le produit est continuellement remis opérationnel par l'intégration continue. Les tests automatisés peuvent être livrés avec le produit pour valider son fonctionnement.

En priorisant les fonctionnalités, chaque livraison du produit répond précisément au besoin le plus important. Chaque itération ajoute quelque chose d'utile au produit. Dépendamment des méthodes, plusieurs variables influencent ce qui doit être priorisé. Le produit est toujours le résultat des décisions prises par le client.

On maximise les efforts investis directement dans la production du logiciel. Il y a peu d'effort perdu sur des éléments inutiles. Les documents utiles ou exigés par le client sont produits. Les efforts logistiques sont limités à leur plus simple expression. Le produit final est un système qui comporte les fonctions essentielles et qui reflète ce qui peut être fait de mieux en fonction de l'investissement [ADM 2005].

2.2.2.3 L'apport agile pour le client

La troisième phrase du manifeste souligne la collaboration avec le client. Dans le cas des projets agiles, la collaboration avec le client est essentielle. Le client est impliqué dans le projet et détermine les priorités. La planification du projet évolue et s'adapte aux changements qu'il propose. Une telle implication amène un changement de vocabulaire. Le client parle alors de « son » système plutôt que du « votre ».

Le client est le principal décideur dans le projet. Le produit est le résultat de ses décisions. La planification récurrente des itérations, évite de gaspiller des efforts dans des fonctionnalités inutiles. Il est responsable de s'assurer que l'investissement comporte réellement une valeur pour lui.

En recevant de petites livraisons rapidement, le client essaie le produit et le valide. Il peut bénéficier rapidement des fonctions essentielles du système et l'exploiter. Il est moins lourd et engageant de procéder de cette manière. Il peut ainsi obtenir un retour sur son investissement plus rapidement. S'il juge que le projet devient trop risqué, il peut décider de le suspendre ou d'y mettre un terme.

Si le produit ne convient pas, il pourra être corrigé. Cette adaptation est un avantage concurrentiel pour le client. On évite l'effet de tunnel et permet de développer les opportunités découvertes au cours du projet. Cette démarche favorise l'émergence des connaissances tacites qui sont difficilement exprimables au départ.

2.2.2.4 L’apport agile pour la gestion de projet

La dernière phrase du manifeste met de l'avant l'adaptabilité. Toutes les méthodes agiles sont inspirées du modèle itératif en spirale de Boehm [Boehm 1988] et par la roue de Deming [Deming 1982]. On ajoute à ce modèle l’aspect incrémental. Toutes les parties du système peuvent être améliorées au cours des itérations. Autrement, le modèle itératif peut devenir une série de cascades.

Les planifications récurrentes sur de courtes échéances, permettent un meilleur suivi et diminuent les risques. Les éléments doivent être complétés à l’intérieur d’un cycle, ce qui évite l’étirement des tâches. L’intégration des étapes de réalisation (conception, codage et tests) contribue à clore les fonctionnalités prévues. Les changements sont introduits lors de ces renouvellements. Au terme du projet, la somme des efforts de planification peut équivaloir à un projet traditionnel, mais réparti différemment à travers le temps.

La révision périodique de l'avancement du projet, peut s'harmoniser avec les suivis budgétaires de l’entreprise et répartir les dépenses suivant les trimestres [Ambler 2004]. Cette révision des ententes partage mieux les risques entre le client et son fournisseur. Le client évite les demandes de changement qui peuvent dépasser le budget. Le fournisseur peut estimer plus précisément, sans avoir besoin de surévaluer pour compenser d’éventuels dépassements [Beauregard 2006]. Cette réduction d’engagement peut faciliter le démarrage d’un projet.

Plusieurs des méthodes agiles ne sont pas complètement structurées et doivent être précisées. L’équipe doit trouver la méthode suffisante pour réaliser le projet, tout en restant dans une zone de confort, qui lui offre une sécurité raisonnable. Avec ces approches, l'équipe a le pouvoir de questionner la pertinence de certaines pratiques. Il s'agit parfois d'analyser un processus pour réaliser que des éléments peuvent être allégés ou éliminés.

2.3 Position de l'agilité

Il est intéressant de positionner l'agilité par rapport à des approches établies. L'alliance agile définit ses exigences via ses valeurs et ses principes. Les autres approches peuvent s'y comparer et déterminer leur conformité. Il est possible de satisfaire différentes approches simultanément. Tout en étant agile, il est possible de se conformer à certaines normes, moyennant certains compromis, donc il est possible d'établir un équilibre satisfaisant.

2.3.1 Conformité aux modèles certifiés

La conformité aux normes préoccupe les entreprises. La certification peut être exigée de certains donneurs d'ordre. Leurs objectifs de prévisibilité ou de répétitivité diffèrent des approches agiles, mais ne sont pas nécessairement incompatibles. L'adaptation d'une méthode agile, pour une certification, demande souvent de compromettre un peu son agilité [Highsmith 2002b; Cockburn 2004; Boehm 2004]. C'est à l'organisation de déterminer ses priorités. Que l'on vise une simple amélioration des processus en s'inspirant d'une norme ou que l'on doive obtenir une certification, les approches agiles ajoutent un ensemble de pratiques intéressantes.

La norme ISO 9001, vise la satisfaction de la clientèle. Ce système de gestion de la qualité a été créé par un besoin de l'industrie, afin de permettre la classification des fournisseurs. Les exigences sont relatives à quatre grands domaines [Wikipedia 2006a] :

- Responsabilité de la Direction : exigences d'actes de la part de la direction en tant qu'acteur premier et permanent de la démarche.
- Système Qualité : exigences administratives permettant la sauvegarde des acquis. Exigence de prise en compte de la notion de système.
- Processus : exigences relatives à l'identification et à la gestion des processus contribuant à la satisfaction des parties intéressées.
- Amélioration continue : exigences de mesure de performance à tous les niveaux utiles, ainsi que l'engagement d'actions de progression.

Il est possible de rejoindre les objectifs de chaque domaine avec les valeurs agiles. Mais, c'est avec le domaine du processus que les organisations peuvent définir des processus lourds, qui limitent l'intégration de l'agilité.

Le CMMi est un modèle visant l'efficience. Il a été créé pour classifier les fournisseurs de logiciel du département de la défense américaine (DoD). Il est composé de cinq niveaux de maturité [Wikipedia 2007] :

1. **Initial** (chaotique): Les facteurs de réussite des projets ne sont pas identifiés, la réussite ne peut donc être répétée.
2. **Piloté** : Les projets sont pilotés individuellement et leurs succès sont répétables.
3. **Standardisé** : Les processus de pilotage des projets sont mis en place au niveau de l'organisation par l'intermédiaire de normes, procédures, outils et méthodes.
4. **Quantifié** : La réussite des projets est quantifiée. Les causes d'écart peuvent être analysées.
5. **Optimisé** : La démarche d'optimisation est continue.

Chaque niveau de ce modèle fait progresser les processus en place, pour être capable de les optimiser. Plusieurs méthodes agiles allèguent équivaloir au niveau standardisé (niveau 3) [Cockburn 2004; ADM 2003]. Nous retrouvons parmi les approches agiles les mêmes principes qui cherchent à améliorer le rendement et la productivité.

Le CMM étant une plate-forme, il comporte un large spectre de modèles de planification acceptables. Le type de planification proposée par les méthodes agiles peut alors cadrer dans le modèle du CMM tel qu'illustré par la figure suivante. L'indiscipline et l'improvisation du « hacking » dans la section gauche contre la planification et la gestion de jalons granulaires à la droite.

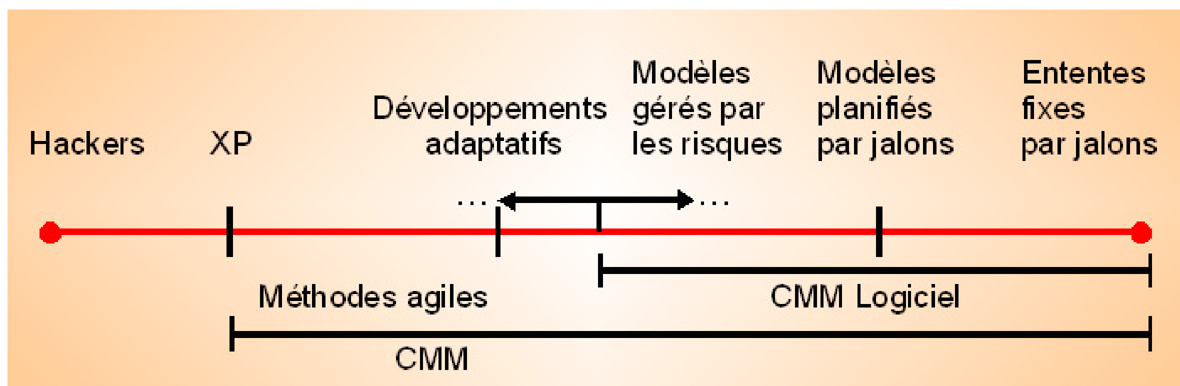


Figure 2-2 : Spectre de la planification.

[Traduit et tiré de Boehm 2002]

Quelle que soit son adaptation, un problème fréquent lorsqu'un processus est défini, est d'imaginer qu'il est figé puisqu'il est rédigé [Cockburn 2004]. Les gens se contraignent à suivre la procédure, sans songer à l'adapter. Parfois, ils préfèrent croire que le processus est figé [Ambler 2004]. La conciliation est possible, mais nous pouvons attribuer cette méconnaissance à la perception plutôt qu'à une contrainte réelle.

2.4 Conformité au modèle agile

Similairement à la conformité aux modèles certifiés, on peut comparer certaines approches par rapport aux exigences agiles. Il est possible qu'une approche soit partiellement agile, car la conformité aux valeurs du manifeste est difficile à quantifier. Il est difficile de quantifier à quel point une équipe collabore bien ensemble, même si elle se voit tous les jours. L'agilité réside dans l'attitude des gens, elle se ressent plus qu'elle ne se mesure. Des recherches sont en cours afin de mieux quantifier le niveau d'agilité [Hartmann 2006]. Il est possible de répondre à une série de questions qui donnent des pistes de réflexion sur le niveau d'intégration des valeurs agiles dans une équipe. [Cockburn 2004]. Nous vérifions ici la conformité de trois approches pouvant être perçue agiles soit le RAD, RUP et les développements libres.

2.4.1 Conformité agile du RAD

Le développement rapide d'application (RAD) est un gabarit de méthode. Il s'appuie principalement sur la réalisation de prototypes pour communiquer les concepts. Ces prototypes servent à discuter avec le client et obtenir son approbation pour déclencher le développement du véritable produit. En analysant les étapes de réalisation suggérées, le RAD ne se qualifie pas comme une méthode agile à cause de son manque d'adaptation aux changements [BI 2001; VTT 2002]. La planification itérative ne vise pas à livrer un produit fonctionnel incrémental. Il s'apparente plutôt à une série de cascades. Il diverge aussi sur le retour d'expérience du client car, elle ne prévoit pas l'intégration de nouvelles demandes.

Des adaptations ont été proposées afin de corriger ces divergences. Les travaux de Vickoff instigateur du RAD en Europe, allaient dans ce sens [Vickoff 2002c]. La méthode DSDM est une adaptation basée sur le RAD, qui propose différents contrôles formant une méthode [DSDM 2005].

2.4.2 Analyse de RUP

Le processus unifié de Rational (RUP) est un gabarit de méthodes qui comporte plusieurs éléments. Sa mise en œuvre sous-entend la réalisation d'une série de livrables approuvés, centrés sur des modèles [BI 2001]. On pourrait y voir une version orientée objets des approches traditionnelles, sans nécessairement être plus efficace [Hesse 2003]. Cette méthode est intimement liée à son outil de conception et au formalisme UML, ce qui diverge des valeurs agiles.

Comme il s'agit d'un gabarit, il est possible d'adapter le RUP. Il peut être aussi bien considéré traditionnel qu'agile. Il est possible de le réduire aux éléments essentiels, afin de se rapprocher d'une méthode agile. Le RUP ne propose pas de guide d'adaptation [VTT 2002]. Il existe toutefois Agile UP, qui est une adaptation proposée par un membre de l'alliance agile [Ambler 2005c].

2.4.3 Analyse des développements libres

Certains classent le développement de logiciel libre (open source) parmi les approches agiles pour leur aspect adaptatif et itératif-incrémental [VTT 2002]. Ces développements partagent plusieurs valeurs avec le monde agile, telles que de planifier de petites livraisons et s'appuyer sur la révision par les pairs pour la qualité du code [Raymond 2003].

Par contre, l'objectif est imprécis, car il est normalement fourni par le client. Les communications entre les développeurs sont plus complexes comparativement aux discussions personne à personne. Les méthodes agiles ont été conçues dans un contexte commercial répondant aux lois du marché. Bien qu'elles partagent des points communs, les développements libres ne sont pas considérés agiles, car ils ne sont pas soumis aux même contraintes. [Cockburn 2001; VTT 2002].

2.4.4 Limitations des approches agiles

Les Agilistes ne prétendent pas que les méthodes agiles sont des « Balles d'argent ». Elles ne sont pas toujours adéquates. Comme ces approches sont encore au stade de découverte, différents essais sont réalisés afin de les éprouver. Il est intéressant de connaître les contextes qui ne tirent pas avantages de ces approches. Ce changement de paradigme n'est pas toujours applicable et ce ne sont pas toutes les entreprises qui ont besoin d'être agiles.

2.4.4.1 Limitations liées au type de projet

Les méthodes agiles cherchent à s'adapter au changement. La situation inverse ne profitera pas d'un tel avantage. Lorsque les spécifications sont stables et bien définies, les modèles traditionnels sont tout à fait convenables. Un environnement technologique bien maîtrisé et stable, diminue l'intérêt d'opter pour une approche agile. On ne tirera pas avantage des rencontres de planification d'itération, qui visent à intégrer les changements.

Une organisation devant être certifiée peut aussi préférer une méthode plus traditionnelle. Les approches agiles comportent moins de « traçabilité », car la production documentaire est secondaire. L'imputabilité et l'irréfutabilité des gestes sont difficilement prouvables. On compromet les traces au profit de la simplicité.

Les projets d'envergures, complexes ou critiques ont aussi intérêt à être pris en charge par une approche plus rigoureuse. La conformité et les validations nécessaires pour la coordination de l'ensemble des composantes sont importantes. Dans un tel contexte, l'autonomie d'un groupe, tel que proposé dans les approches agiles, constitue un risque.

Souvent liées à l'envergure d'un système, de grosses équipes de développement demandent plus de coordination. On doit établir un plan de communication et passer par des voies officielles. Cette complexité à communiquer est aussi observable dans des équipes physiquement réparties. Une telle complexité à communiquer limite l'agilité.

2.4.4.2 Limitations liées à la culture organisationnelle

Une autre source de limitation est liée aux facteurs humains. Les gens ont peur de l'inconnu qui menace leur confort et ils résistent aux changements [Cockburn 2004]. Les gestionnaires et autres groupes impactés, doivent aussi être conciliants à ces changements.

Les approches agiles sont synonymes de changement. Les gens anticipent le dérangement et craignent la perte du confort acquis avec les procédures existantes. Les approches agiles touchent à la fois les aspects d'origine personnelle, procédurale et culturelle, ce qui les rend difficile d'implantation.

Les approches traditionnelles sont prédictives, voire proactives. Les approches agiles sont adaptatives, voire réactives. Nous opposons proactif et réactif comme l'expression : «Mieux vaut prévenir que guérir ». Ce préjugé favorise la planification, puisqu'on y trouve un aspect rassurant en prévoyant la marche à suivre. Proposer de s'adapter peut alors inquiéter et laisser croire qu'on ne sait pas où l'on va.

L'appui de la direction est un facteur important dans l'introduction de changements. Les approches agiles fonctionnent avec des équipes autonomes autorisées à prendre des décisions. Le gestionnaire doit laisser plus de contrôle à l'équipe de développement, ce changement de culture n'est pas toujours souhaité. Le gestionnaire en charge du projet doit exercer un leadership, avoir confiance en son équipe et rester solidaire.

Le gestionnaire doit aussi s'efforcer de mieux connaître les membres de son équipe et connaître leurs forces et leurs faiblesses. Le développeur n'est pas une ressource linéaire dont le comportement est prévisible [Cockburn 2000]. Les différentes compétences les distinguent, ce qui diverge de la vision que cette ressource puisse être interchangeable. Cette notion complexifie le travail d'un gestionnaire, qui préfère considérer que les gens sont comme le temps et qu'ils sont interchangeables [Brook 1975]. La planification est ainsi plus difficile à produire, car on doit tenir compte des compétences individuelles plutôt que des rôles.

2.4.5 Les supports agiles

Il existe divers moyens pour s'informer et être guidé dans l'univers agile. Cette section décrit les différentes ressources disponibles pour s'initier et suivre les événements de cette communauté.

2.4.5.1 La communauté

Le meilleur endroit pour débuter son initiation avec les approches agiles est le site de l'alliance (www.agilealliance.org). Depuis la publication du manifeste, l'agilité a trouvé plusieurs adeptes. L'alliance agile est un organisme à but non lucratif. Plusieurs membres fondateurs y sont impliqués, on compte parmi eux plusieurs inventeurs de méthodes. Il est possible pour un individu ou une corporation de devenir membre. Le site contient d'excellentes références sur différents sujets agiles.

Souvent associés, différents sous-groupes nationaux ou groupes d'utilisateurs régionaux partagent l'objectif de rayonnement des ces approches. C'est le cas du réseau Agile Canadien (www.agilenetwork.ca) et plus près de nous il y a Agile Montréal (www.agilemontreal.ca) et Agile Québec (www.agilequebec.ca). Ces organismes proposent différentes activités de sensibilisation afin de partager les expériences et faire connaître ces approches.

2.4.5.2 Les outils

Bien que ce ne soit pas une priorité pour les Agilistes, ils utilisent plusieurs outils. Que ce soit pour supporter le développement, les rencontres ou l'apprentissage, ils facilitent tous l'intégration des pratiques agiles.

Une première série d'outils est simplement composée de matériel de bureau (papier, crayons, tableau blanc). Ils sont dédiés à développer la qualité des communications et faciliter les rencontres de travail. On propose de les exploiter de manière originale pour en tirer avantage. Par exemple, l'utilisation de tableau blanc permet à tous de participer en y allant de son schéma en utilisant différentes couleurs et des pictogrammes simples. L'utilisation de papiers pré-collés (post-it) est aussi utile lors des discussions, afin d'ordonner un processus sur un mur ou un tableau. Ils peuvent être collés, déplacés et

même superposés les uns sur les autres. Pour conserver en archive les résultats d'une rencontre, une simple photo numérique suffit et personne ne perd son temps à retranscrire. L'utilisation d'outil aussi simple, comporte l'avantage d'impliquer les gens activement. Ils sont moins timides et s'expriment plus librement afin de maximiser la qualité de la rencontre.

En plus des outils conventionnels, plusieurs outils logiciels sont apparus pour supporter les pratiques agiles. Principalement au niveau de l'assurance qualité. Le très populaire J-Unit (et toutes les versions xUnit) est un incontournable pour l'automatisation des tests unitaires. On trouve aussi des produits pour les essais fonctionnels et les compilations automatisées. La popularité des intranets, des logiciels de gestion de contenu et des « Wiki », simplifie la diffusion d'information rapidement. Plusieurs de ces outils sont issus du monde du logiciel libre, ce qui les rend faciles d'essais et peu engageants. Une liste de ces outils est disponible sur le site de l'alliance.

Une dernière série d'outils à vocation pédagogique est aussi disponible. On peut inclure des groupes de discussions, des publications et aussi des jeux qui simulent des situations qui faciliteront les rencontres ou permettront d'expérimenter des pratiques agiles.

2.4.5.3 Adoption par l'industrie

Une autre forme de support aux approches agiles est l'adhésion des entreprises. Sur le plan commercial, l'agilité est un argument de vente pour les firmes de services et les grandes corporations. Présentement, nous voyons le terme repris dans les campagnes de marketing. Plusieurs grandes corporations, telles que Microsoft, IBM, Borland et Oracle ont déjà montré des signes d'adoption de l'agilité. Les outils qu'ils produisent, favorisent de plus en plus l'automatisation des tests et le « refactoring ». Cette amélioration constante des outils favorise l'intégration de ces pratiques.

2.5 Distinction entre les approches traditionnelles et agiles

Les méthodes agiles ont été développées en réponse à l'inefficacité et à la lourdeur des approches traditionnelles [Fowler 2005]. Ces projets sont planifiés minutieusement et visent à prédire leurs déroulements. À l'opposé, les approches agiles préfèrent s'adapter au cours du projet.

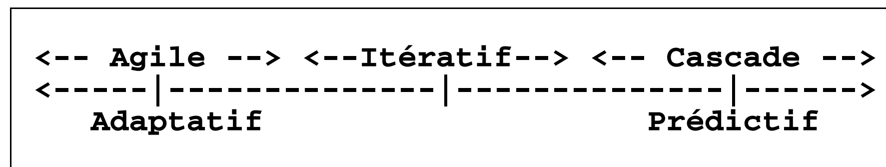


Figure 2-3 : Continuum adaptatif à prédictif

[Tiré et traduit de Boehm 2004]

En imaginant un continuum (voir Figure 0-2) partant de l'adaptatif jusqu'au prédictif, nous pouvons établir que les approches agiles sont du côté adaptatif et que les méthodes traditionnelles sont du côté prédictif. Bien que cet aspect les éloigne, elles ne s'opposent pas sur tous les plans. Nous pouvons grouper les propriétés communes aux deux approches et celles qui leurs sont spécifiques.

2.5.1 Propriétés communes et spécifiques

Les approches agiles et traditionnelles se rejoignent sur différents aspects. Les propriétés communes et les propriétés spécifiques, qui caractérisent les deux types d'approches, sont présentées à travers différents tableaux qui touchent un ensemble de caractéristiques. [Levine 2005; ADM 1996a; VTT 2002; Boehm 2002].

Caractéristiques des équipes de développement		
	Traditionnelle	**Agile**
Propriétés communes	Apprendre par l'expérience. Formation et expérimentation. Intégration de nouvelles technologies.	
Grosseur des équipes	Peut aller au-delà de 20 développeurs.	Moins de 12 développeurs.
Séparation des tâches	Liées à la fonction ou à la spécialité, en fonction du livrable à produire.	Liées à l'intérêt ou à la compétence, s'appuyant sur la collaboration de l'équipe.
Communication des spécifications	Par écrit. Définit selon la méthode et retraçable.	Par des échanges verbaux, journaliers et en personnes.
Localisation des équipes	Bureaux séparés. Équipes distribuées.	Préférablement dans une aire commune.
Profil des compétences de l'équipe	Équipe multidisciplinaire. Accès à des spécialistes. Débutant. Encadré.	Personnel compétent. Autonome. Discipliné. Partage de connaissances.

Tableau 2-4 : Caractéristiques des équipes de développement

Caractéristiques liées au produit		
	Traditionnelle	**Agile**
Propriétés communes	Établir les spécifications fonctionnelles. Production de la documentation utile. Rencontrer les exigences de qualité.	
Architecture du système	Architecture optimisée en prévision des plans.	Architecture pour les besoins immédiats et émergeants.
Spécifications fonctionnelles	Rigoureuses.	Spécifications ouvertes répondants à l'objectif.
Techniques particulières de développement	Réutilisation de classes. Couplage de composants. Assemblage par étape.	Automatisation des tests. Refonte du code (refactoring).

Tableau 2-5 : Caractéristiques liées au produit

Caractéristiques de la gestion de projet		
	Traditionnelle	**Agile**
Propriétés communes	Planification du projet. Études préliminaires de faisabilité. Gestion des coûts et reddition de comptes.	
Facteur de succès	Finir en temps et dans le budget.	Fournir une valeur ajoutée.
Type de projet	Besoins connus et stables. Incidence critique. Grand projet.	Besoins changeants. Incidence peu critique. Livraison rapide.
Modèles du processus	Modèles variés (cascade, v, etc.) visant à prédire le déroulement. Suit toujours l'ordre : Designer – coder – tester.	Modèle itératif incrémental. Adaptable selon les pratiques et l'environnement. Réalise simultanément : Designer – coder – tester.
Planification	Détaillée pour l'ensemble du projet. Diagramme de Gantt élaborant les affectations de tâches à travers le temps.	Grossière pour l'ensemble, détaillée pour l'itération. Production d'une liste de fonctionnalités et de tâches dans un tableau décroissant (burn chart).
Suivi de la progression	Respect de la planification. Approbation des livrables (dossiers, modèles, documentations).	Respect des fonctionnalités. Présentation fonctionnelle du logiciel.
Gestion du risque	Suivi plus rigoureux.	Courtes itérations favorisant le plus grand risque au départ.
Gestion des changements	Préfère les minimiser. Limitée aux éléments les plus valables, car elles sont souvent coûteuses.	Favorable à leur intégration. Intégrée au plan d'une itération.
Délais de livraison	À la fin du projet ou dès la première itération, pouvant dépasser 6 mois.	De deux semaines à quatre mois. Généralement un mois.

Tableau 2-6 : Caractéristiques de la gestion de projet

Caractéristiques de l'organisation		
	Traditionnelle	**Agile**
Propriétés communes	Comprendre les besoins d'affaires. Implication des utilisateurs. Satisfaction du client.	
Culture organisationnelle	Recherche l'ordre et la standardisation. Adéquation aux règles de l'industrie. Similaire au génie traditionnel. Commande et contrôle. Suit les processus établis. Recherche une certaine assurance. La production du logiciel est une activité de construction, qu'il est possible de planifier en détail.	Sur le bord du chaos, avant-gardiste. Place à la créativité et à l'innovation. Collaboration et leadership. L'organisation doit faire confiance et vivre avec les décisions des développeurs. La production du logiciel est une activité créative, qui est impossible de planifier en détail.
Participation avec le client	Entente contractuelle pouvant être forfaitaire. Peut se déresponsabiliser du système. Participe aux spécifications et aux approbations. Surtout sollicité au début et à la fin du processus.	Tarif horaire, révisions mensuelles. Décide du contenu. Sentiment de responsabilisation envers le système. Demande une forte implication. Participe aux spécifications et doit être disponible tout au cours du projet. Approuve le produit et fournit ses commentaires pour apporter les corrections.

Tableau 2-7 : Caractéristiques de l'organisation

Les deux approches cherchent à satisfaire les besoins du client, en trouvant les meilleures solutions possibles. La différence est au niveau des moyens utilisés pour y arriver. Il n'est pas nécessaire de suivre complètement une méthode. L'organisation doit trouver son d'équilibre en fonction de ses priorités, sa tolérance aux risques et son efficacité.

2.6 Conclusion

L'agilité se définit dans le domaine logiciel par sa capacité d'adaptation et son attention aux individus. Le manifeste intègre l'ensemble des valeurs priorisées par ce mouvement de pensée. L'agilité est un changement de paradigme, qui impacte sur la culture organisationnelle. Elle représente un défi, car elle implique à la fois les développeurs, les gestionnaires et les clients. Ultimement, elles visent à satisfaire le client par un produit fonctionnel adéquat aux besoins réels.

Différents intervenants tirent des avantages de ces approches. Comparativement aux approches traditionnelles, les approches de type agile veillent à ce que les efforts investis maximisent le bénéfice du client ; que le produit soit adapté aux besoins qui seront découverts au cours du développement ; que les gens impliqués dans le projet soient motivés et heureux d'y participer ; et finalement que le projet soit bien suivi pour limiter les risques.

Les approches agiles viennent élargir l'offre méthodologique. Elles émergent de l'évolution du domaine logiciel qui précise son identité. Elles s'harmonisent avec l'amélioration des technologies et l'accessibilité du domaine au grand public. Les organisations n'ont pas toutes besoin d'être agiles. Par contre, elles ont toutes besoin de savoir quelle est l'offre des approches agiles, afin qu'elles puissent se positionner.

Lorsque le contexte est favorable à son implantation, l'agilité apporte plusieurs améliorations. L'organisation sait qu'elle suit un modèle à sa mesure et évolutif. Alors que certains modèles certifiés cherchent à optimiser vos opérations selon votre passé, les méthodes agiles vous préparent à vous adapter à votre avenir.

Lorsqu'on discute des limitations des approches agiles, les gens préfèrent souvent conclure qu'elles ne seront pas adéquates pour eux. Il est tout de même possible de trouver des compromis viables entre deux paradigmes. Il est possible de balancer les visions agiles à l'intérieur d'une structure plus traditionnelle. Nous sommes présentement à l'étape d'intégrer et d'accepter ces approches comme des alternatives possibles. Plusieurs mouvements du marché expriment cet état.

Les approches agiles témoignent d’une évolution du domaine répondant aux pressions du marché. Comme dans plusieurs autres domaines, elles visent à être plus efficaces avec moins de ressources. Elles partagent des propriétés communes héritées des méthodes établies, mais comportent aussi plusieurs distinctions qui les caractérisent. En comparant avec les approches traditionnelles, nous constatons que certains aspects se recoupent et qu’il est possible d‘équilibrer les pratiques.

Le prochain chapitre décrit une série de méthodes agiles. Elles traduisent les valeurs agiles du manifeste, par des pratiques. Chacune des méthodes comporte ses particularités qui peuvent s’avérer intéressantes pour les organisations qui partagent les mêmes préoccupations.

Chapitre 3: Description de méthodes agiles

La plupart des méthodes agiles ont été élaborées et proposées par des praticiens œuvrant en consultation. Ces méthodes étant relativement jeunes, les références sont assez limitées. Ce chapitre offre un tour d'horizon de différentes méthodes agiles existantes.

Trois d'entre elles sont détaillées : Extreme Programming (XP) pour sa popularité et ses pratiques de programmation ; SCRUM pour sa popularité grandissante et ses pratiques de gestion de projets ; Crystal Clear pour ses valeurs et sa versatilité. Chaque description comporte un bref historique, expliquant les motivations de base des inventeurs et la description de la méthode. Les annexes A, B et C comportent les observations apportant des éléments d'analyse et des critiques. On complète avec un survol de quelques autres méthodes de l'alliance agile.

3.1 Extreme Programming

Extreme Programming (XP) est une méthode agile qui priorise l'excellence technique. Elle comporte un ensemble de pratiques interdépendantes qui demandent beaucoup de rigueur, afin de puiser le maximum d'efficacité des équipes de développement.

3.1.1 Origines de XP

3.1.1.1 Son inventeur

Extreme Programming (XP) a été développé par Kent Beck, Ward Cunningham (inventeur du Wiki) et Ron Jefferies au cours des années 1990. Beck est le principal instigateur de XP. Tout en s'intéressant à plusieurs sujets avant-gardistes et au fil des ans, il collabora avec Erich Gamma (1991) et Grady Booch (1993) [Beck 2005]. Le plus important dans la vision de Beck est l'interaction sociale, afin de changer la manière dont les gens se traitent les uns les autres et comment ils sont traités par l'organisation. XP prône les valeurs de communication, la simplicité, le feedback et le courage. Elle est la plus intéressante des approches agiles [Highsmith 2002a]. Elle se concentre sur le client, le gestionnaire et le programmeur. Elle accorde aussi des droits et des responsabilités à ceux qui jouent ces rôles [Beck 2000].

3.1.1.2 Les racines de XP

Beck n'a pas inventé les pratiques de XP et il ne s'en cache pas. Ce qu'il a inventé, c'est la synergie qui les relie. Il s'est inspiré de diverses sources qui ont servi à bâtir cette méthode [Beck 1999].

Cette méthode est vouée aux environnements, dont les spécifications changent fréquemment, entraînant une évolution rapide de la planification du projet. On part d'un petit système qui évoluera et se précisera au cours de son développement, ce qui rappelle le modèle en spirale. Elle marque une scission entre les décisions d'affaires et techniques. On construit les spécifications et la planification selon une perspective des caractéristiques. L'utilisation des métaphores aide à synthétiser les concepts.

Cette méthode précise aussi l'attitude des participants et l'aménagement des espaces de bureaux issues d'études sur le sujet. L'annexe F (Les racines de XP) présente la section de l'article de Beck qui répertorie ses sources.

3.1.2 Description de la méthode

Cette méthode a été créée pour répondre aux changements émergeants de l'apprentissage des clients en cours de projet. Elle s'adresse principalement aux petites équipes composées de 2 à 12 personnes, travaillant dans un espace commun. Elle demande une grande implication du client et exige de pouvoir automatiser les tests. Elle augmente la productivité et offre de meilleurs résultats. Son objectif consiste à livrer le logiciel requis au temps requis [Wells 1999]. Elle se distingue avec ses pratiques particulières et ses valeurs. Elle est exigeante, car chaque pratique en renforce une autre, ce qui forme un tout cohérent. Si vous n'appliquez pas les toutes les pratiques, vous ne suivez pas XP [Beck 2000].

3.1.2.1 Les valeurs de XP

XP repose sur 4 valeurs fondamentales [Beck 2000] :

- **La communication** : C'est le moyen fondamental d'éviter les erreurs. Le moyen à privilégier est la conversation directe, face à face. Les moyens écrits ne sont que des supports et des moyens de mémorisation.

- **Le courage** : Il est nécessaire à tous les niveaux et de la part de tous les intervenants, notamment chez les développeurs (lorsque des changements surviennent à un stade avancé du projet ou quand des défauts émergent) et chez le client (qui doit pouvoir prioriser ses demandes).
- **Le retour d'information** (*feedback*) : Les itérations sont basées sur les retours d'informations du client, permettant une adéquation totale entre l'application finale et sa demande.
- **La simplicité** : Avec XP, la façon la plus simple d'arriver à un résultat est la meilleure. Prévoir préalablement des évolutions futures n'a pas de sens. Ce principe est résumé en une phrase : « Tu n'en auras pas besoin. » (en anglais « *You ain't gonna need it.* »). La meilleure manière de rendre une application extensible est de la garder simple (et donc simple à comprendre) et bien conçue autant que possible.

3.1.2.2 Les pratiques

Ses valeurs se déclinent en 13 pratiques. Ces pratiques se regroupent en quatre grandes catégories qui touchent la planification, la conception, la réalisation et les tests [Beck 2000; Wells 1999]. Il est possible de décomposer ces pratiques en couches, pour faciliter leur implantation. Consulter l'annexe A (Observations sur XP) pour avoir un exemple de regroupement des pratiques. Ces pratiques ne sont pas nouvelles, ce qui est nouveau c'est la manière dont elles sont appliquées [Highsmith 2002a].

Pratique de planification

- **Séance de planification (the planning game)**: Conjointement avec le client, l'équipe créer des scénarios d'utilisation (user stories), ce qui fournit l'ensemble des fonctionnalités attendues du système. Cette étape revient à chaque itération. Le projet est considéré comme achevé, quand le client n'est plus en mesure de trouver de nouveau scénario.

Pratiques de conception

- **Conception simple** : Chaque élément doit être conçu le plus simplement possible, ce qui rend le code simple à comprendre et facile à modifier. Cette notion évite de créer des éléments spéculatifs qui ne serviront probablement jamais [Highsmith 2002a].
- **Utilisation de métaphores** : On utilise des métaphores et des analogies pour décrire le système et son fonctionnement, ce qui permet de le rendre compréhensible par les non-informaticiens. Ces références de haut niveau ne remplacent pas l'architecture du système, mais synthétisent une vision équivalente [Highsmith 2002a].
- **Réfection du code** : Amélioration continue de la qualité du code sans en modifier le comportement (commentaire du code, respect des standards, etc.).

Pratiques de réalisation

- Un représentant du **client sur place** : Afin d'assurer une meilleure réactivité, un représentant du client doit être présent pendant toute la durée du projet. Cet utilisateur sert de référence pour l'équipe.
- **Intégration continue** : Tout le produit est assemblé à chaque fois qu'une tâche est terminée, ce qui permet d'avoir toujours une version à jour, notamment pour la livraison ou pour le démarrage des nouvelles tâches.
- **Livraisons fréquentes** : Les livraisons doivent être les plus fréquentes possibles, afin d'obtenir un feed-back rapide, tout en offrant des fonctionnalités complètes.
- **Rythme soutenable** : Il faut éviter les heures supplémentaires. Dans tel cas, il faut redéfinir le planning. L'uniformité du temps investi est important dans le calcul de la vélocité de l'équipe. La relation entre les estimés et le rendement est directement lié.
- **Standard de code** : Le code est une propriété collective. Les standards facilitent la communication et la rapidité de compréhension.
- **Programmation en binôme** : La programmation se fait par deux. Les développeurs changent fréquemment de partenaires, ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.
- **Appropriation collective du code** : L'équipe est collectivement responsable de l'application et est supposée connaître l'intégralité du code. Chacun peut modifier toutes les portions du code.

Pratiques de tests

- **Tests de recette** : À partir de critères définis par le client, on créer des procédures de test, souvent automatisées, qui permettent de vérifier fréquemment le bon fonctionnement de l'application.
- **Tests unitaires** : Un test unitaire est développé avant d'implémenter une fonctionnalité. Ces tests découlent des recettes de plus haut niveau.

3.1.2.3 Processus d'extreme Programming

Afin de visualiser le processus, nous pouvons schématiser les pratiques dans leur contexte. Nous pourrons voir les détails sur différentes échelles : au niveau du projet, des itérations, du développement et de la propriété commune.

Au niveau du projet

Extreme Programming Project

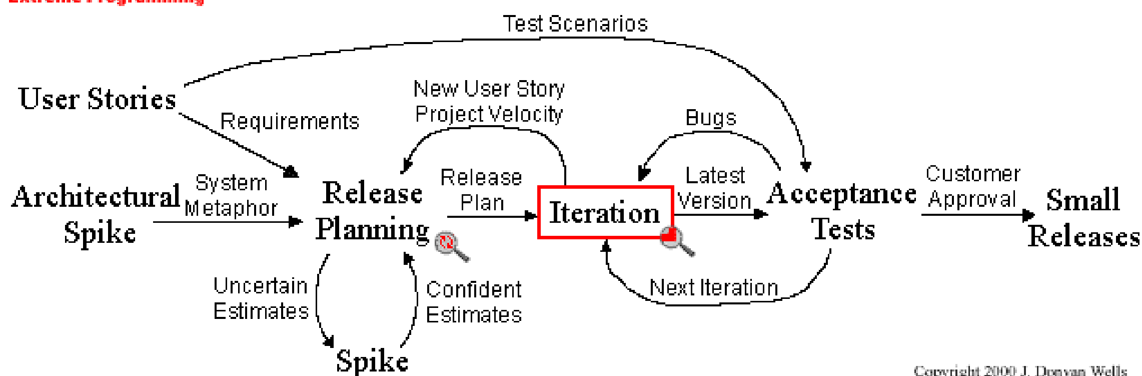


Figure 3-4 : Processus XP au niveau du projet

Au niveau du projet, les **scénarios utilisateurs** sont les principaux éléments qui mènent le projet. Ils sont composés par le client qui exprime ses besoins. Ils fournissent la liste des caractéristiques à réaliser et serviront de référence lors des tests d'acceptation.

Les **transitions architecturales** (architectural spike) forment les grandes parties du système qui construiront les métaphores du projet. L'utilisation de métaphores aide à la communication et permet de regrouper les scénarios ensemble.

La **planification de la livraison** liste les scénarios qui seront réalisées au cours des itérations. Cette planification se construit en fonction des priorités du client, des estimations et de la vélocité de l'équipe qui prévoit être en mesure de produire ces scénarios pour cette itération.

Si une incertitude survient sur une estimation, une **transition** (spike) amène un peu plus de rigueur à cette estimation. Elle sert de preuve de concept, afin de mieux évaluer la charge de travail, ce qui élimine les solutions infructueuses. Si quelque chose ne fonctionne pas, il est préférable de le savoir rapidement.

Lorsque la liste des scénarios à réaliser est complétée, les cycles des **itérations** pour cette livraison commencent. La durée des itérations doit être constante. Une durée de deux à quatre semaines est conseillée. Cette constance aide lors de la planification, puisqu'elle sert

au calcul de la vélocité de l'équipe. La vélocité est une mesure estimant la charge de travail qu'une équipe peut réaliser en un laps de temps. Les fonctionnalités étant estimées selon une charge de travail, l'équipe peut déterminer combien de fonctionnalités elle peut livrer lors de cette itération. Cette notion, similaire aux estimations en jour/personne, se raffine à chaque itération qui permet aux équipes de toujours améliorer leur exactitude.

Les **tests d'acceptation** suivent les itérations. Ces tests d'ordre fonctionnel, comparent les fonctionnalités réalisées aux scénarios décrits préalablement pour en vérifier la conformité. Les problèmes soulevés sont alors complétés dans une version suivante. Ces conformités sont vérifiées par le client.

Lorsque satisfaisante, elle sera une nouvelle **petite version**. Suite à la livraison de toutes ces petites versions, le projet est terminé.

Au niveau de l'itération

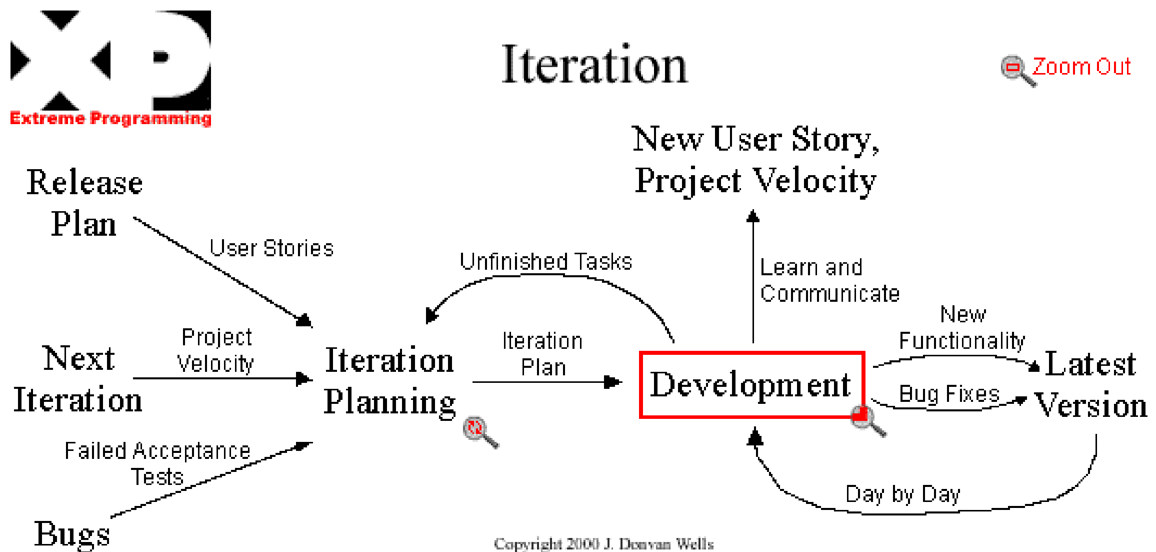


Figure 3-5 : Processus XP au niveau de l'itération

L'itération est un mini projet en soit. Les intrants de cette phase sont la liste des scénarios utilisateurs à réaliser lors de cette itération et les « bugs » rencontrés aux itérations précédentes. En fonction de sa vélocité, l'équipe estime ce qu'elle pourra livrer.

Ces différents éléments sont consolidés, afin de composer le **plan d'itération**. Les scénarios non-inclus seront réalisés ultérieurement dans les prochaines itérations de la version.

Lorsque la liste est complétée, l'équipe entre dans la phase de **développement**. Même au cours de cette activité, l'équipe peut débusquer de nouveaux scénarios. Cette phase est riche en interaction avec le client. Le cycle d'apprentissage s'opère naturellement.

Chacune des corrections et des fonctionnalités complétées, sont intégrées tous les jours à la **dernière version**. De cette manière, le produit progresse vers son objectif. Le client peut constater concrètement les efforts de l'équipe et voir son système opérer.

Au niveau du développement

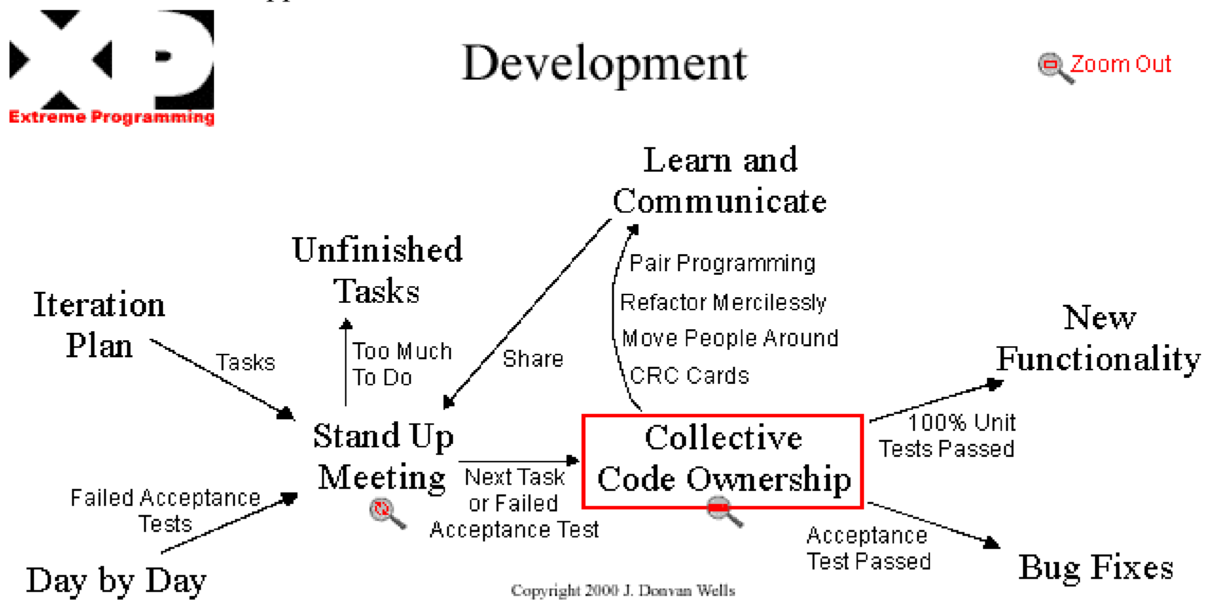


Figure 3-6 : Processus XP au niveau du développement

L'activité principale d'XP reste le développement. Le quotidien des développeurs commence par la **rencontre matinale** (Stand Up Meeting) qui passe en revue les tâches à réaliser suivant le plan d'itération, ainsi que les tâches précédentes incomplètes. C'est aussi lors de cette rencontre que les demandes trop exigeantes sont mises de côté dans la liste des **choses à compléter**.

La **propriété commune du code** fait appel à toutes les compétences de l'équipe même. C'est la pierre d'assise d'XP, elle est expliquée plus en détail à la page suivante. Chaque développeur apporte ses connaissances au projet, afin de partager les éléments qui peuvent aider à la réalisation des fonctionnalités.

Il en ressort de **nouvelles fonctionnalités** qui réussissent entièrement leur suite de tests unitaires. La **correction des erreurs** est aussi validée au cours de cette pratique.

La propriété du code commun

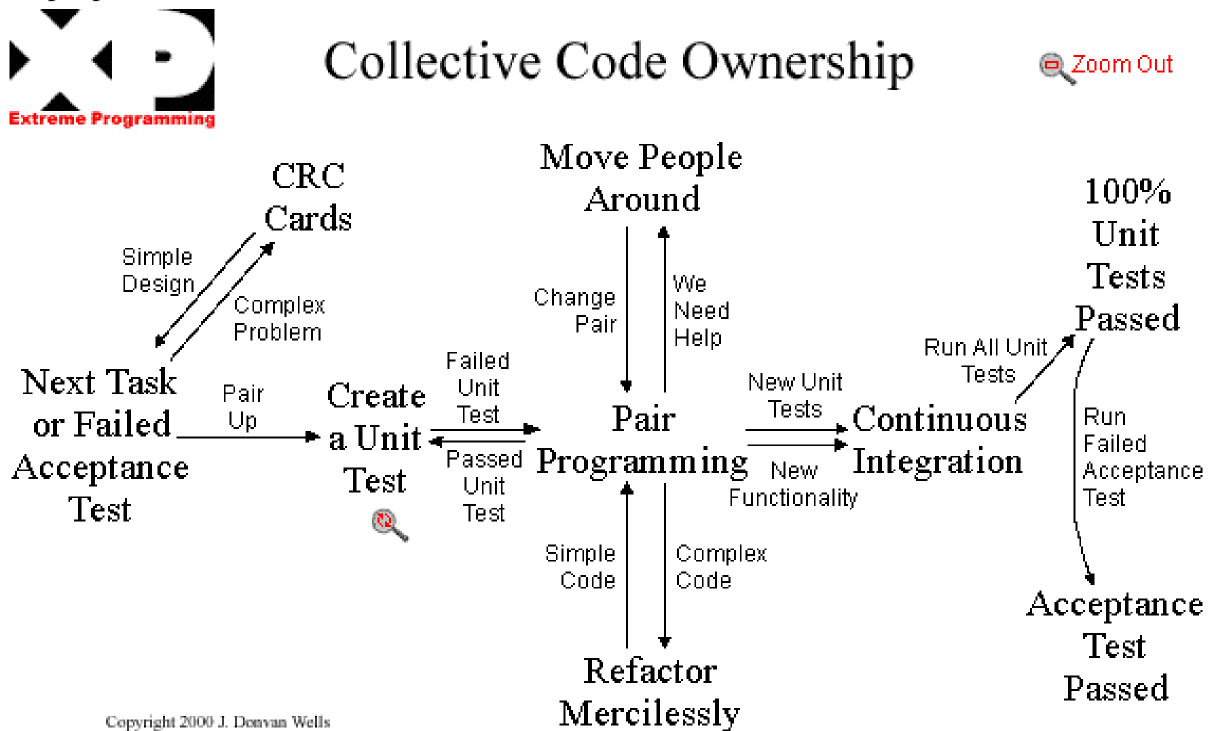


Figure 3-7 : Processus XP sur la propriété du code commun

La pratique de la propriété du code commun, est en fait la réalisation informatique des scénarios utilisateurs décrits sur les **cartes CRC** (Classe, Responsabilité et Collaboration donc, en anglais : CRC Cards). Le design et la réalisation sont faits conjointement.

Les développeurs prennent connaissance de la tâche à réaliser. Cette tâche peut être nouvelle ou provenir d'une précédente ayant échoué son test d'acceptation que le client ne jugeait pas adéquat. Les développeurs commencent par préparer les tests unitaires qui leur permettront de vérifier le résultat de la fonction qu'ils produiront.

La **programmation** s'effectue en pairs (pair Programming). Les gens créent, corrigent et simplifient le code source. Le code doit suivre les standards (nomenclature, commentaire etc.) et il doit être conçu de la manière la plus simple et compréhensible possible. Le code doit se limiter à être capable de réaliser les cas de tests que l'on attend de lui. De nouveaux cas de tests unitaires peuvent aussi s'ajouter, afin de colmater les autres erreurs possibles nouvellement décelées.

C'est autour de cette activité que les gens peuvent consulter et **perfectionner** le code existant, code qu'ils n'ont pas nécessairement fait eux-mêmes. Les tests unitaires automatisés viendront confirmer que tout fonctionne encore correctement après ces modifications.

Si une équipe requiert de l'aide, la paire peut être changée. Cette « **circulation** » (move people around) renforce les expertises de chaque développeur. L'apprentissage est constant et diversifié. C'est par cette pratique que la connaissance du système se propage, ce qui remplace la lecture des spécifications écrites.

Tout le code et les tests unitaires liés sont intégrés continuellement sur un poste dédié à cette fin. Ce poste est la référence qui garanti l'intégrité du produit. Il assure que les tests unitaires passent dans le contexte du système. Si une régression est observée, elle sera corrigée. Si la correction ne peut être fournie immédiatement, l'ajout est retiré et le système est remis dans son état précédent. C'est une forme de contrôle de configuration du produit garanti en tout temps, une version fonctionnelle de tout le système.

À chaque jour, tous les tests unitaires qui fonctionnent sont soumis au **test d'acceptation** (acceptance test passed) par le client qui est disponible sur les lieux. Les tests d'acceptation sont validés avec les cartes CRC, qui sont les aide-mémoire du projet. Si une fonctionnalité du système ne correspond pas aux attentes du client, il peut la clarifier directement auprès des développeurs qui arriveront à mieux comprendre ce qu'ils doivent réaliser. La progression du nombre de tests acceptés par itérations, est illustrée par la figure suivante [Tiré de Jefferies 1999].

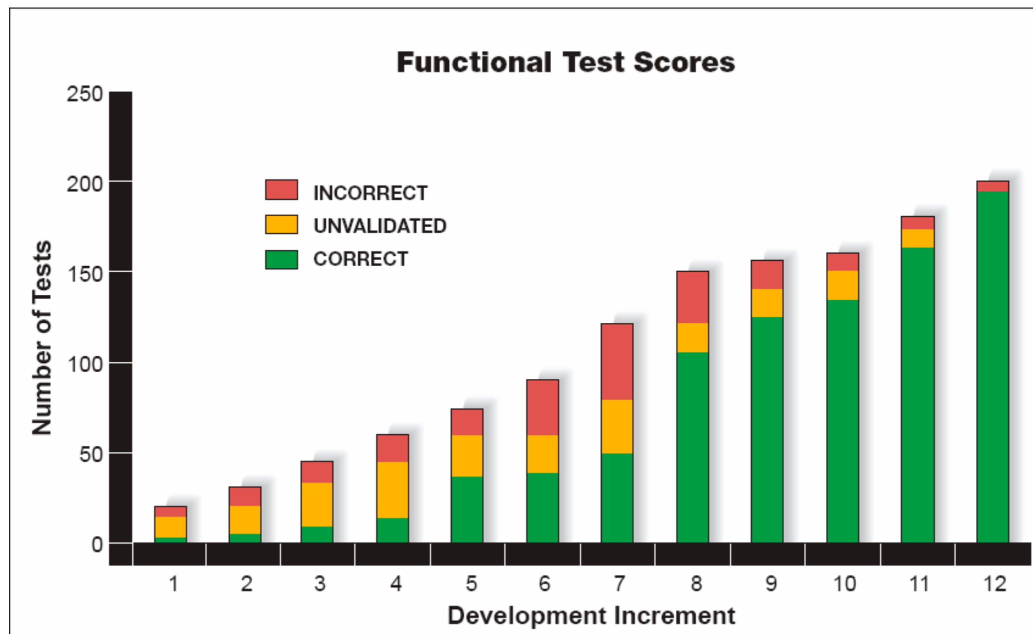


Figure 3-8 : Progression des tests acceptés par itérations

Séquence des activités

XP remet en question la séquence des tâches préconisées par les méthodes en cascade : Analyse – Design – Implémentation – Test. Cette méthode diffère aussi des méthodes itératives qui séparent le projet en plus petites phases, mais qui obéit à la même séquence d'opérations. Dans XP, après une première phase couvrant l'ensemble des fonctionnalités du système, toutes les autres tâches sont réalisées de manière parallèle, couvrant ainsi toutes les fonctionnalités et permettant rapidement d'altérer le design.

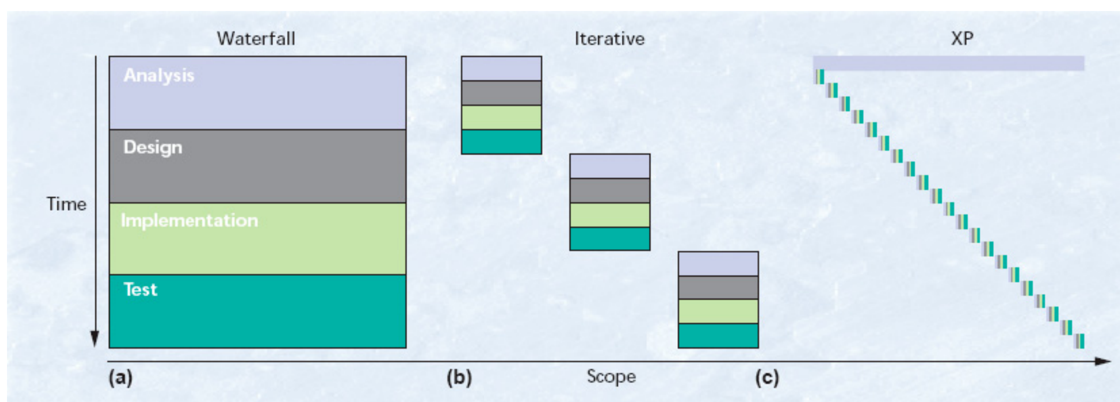


Figure 3-9 : Séquence des activités selon différentes méthodologies.

[Tiré de Beck 1999]

Cette manière de procéder met en relief deux choses. Premièrement, l'équipe est autorisée à prendre des décisions sur le design en même temps qu'elle teste et programme le logiciel. Deuxièmement, le développement ne cloisonne pas les gens dans une spécialité comme l'architecture, l'analyse fonctionnelle, la programmation et l'assurance qualité. L'équipe aborde le projet en silo. Elle fragmente le travail et pousse une petite partie jusqu'au bout, avant d'entreprendre une autre section. Cette manière de procéder développe la compétence de l'équipe sur tous les niveaux. Cette pratique réalise plus rapidement des éléments que le client pourra utiliser, avant que le système ne soit réellement complet. Ainsi, le client pourra bénéficier plus rapidement d'un retour sur l'investissement.

La dynamique créée par ce cycle d'interdépendance, amène les clients et les développeurs à mieux se connaître et à mieux communiquer. Chaque participant au projet travaille pour ajouter plus de valeur au produit. Les développeurs et les clients améliorent ainsi leur expertise dans leurs estimations, ce qui crée un cycle d'apprentissage.

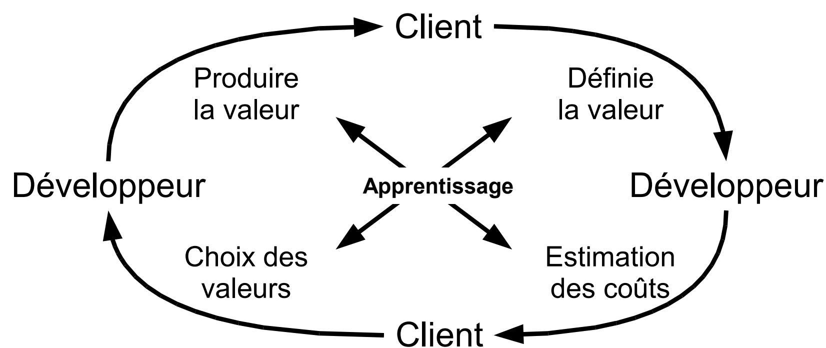


Figure 3-10 : Apprentissage de l'estimation de la valeur et des coûts.

[Traduit et tiré de Beck 2000]

Pour en connaître davantage, l'annexe A (Observations sur XP) présente les différentes analyses et critiques sur cette méthode.

3.1.3 Conclusion sur XP

Elle est la plus connue des méthodes agiles donnant ainsi l'illusion qu'elle est la seule. Son côté « extrême » dérange. Bien définie par ses pratiques, elle s'accroche surtout aux quatre valeurs fondamentales qui valorisent la communication et les relations humaines. Elle insiste sur la confiance et la qualité, deux éléments importants pour plusieurs développeurs.

La méthode XP est exigeante et ne convient pas à toutes les organisations, elle a une niche particulière, dont les limites ne sont pas encore toutes tracées. Elle s'adresse à de petites équipes autonomes et expérimentées, travaillant dans un contexte où ils ont à découvrir progressivement, avec l'aide du client, le système à construire. C'est une approche qui mérite d'être connue, afin de proposer une vision différente.

3.2 SCRUM

SCRUM est un processus de contrôle empirique de développement logiciel. Il permet aux équipes de produire des logiciels de manière itérative incrémentale, favorisant ce qui a le plus de valeur pour l'entreprise [Norton 2005].

3.2.1 Origines de SCRUM

3.2.1.1 Son inventeur

Ken Schwaber est un développeur d'expérience, président de la firme de consultant Advanced Development Methods, Inc (ADM). Cette firme, dédiée à l'amélioration de méthodologies, développa un outil de support nommé MATE (Methods And Tools Expert). Elle a été parmi les pionnières en matière d'outils de gestion des processus. Cet outil était implanté dans plusieurs grandes entreprises. Son cheminement dans ce domaine l'amena à être un partisan évident des procédés adaptatifs, légers et empiriques pour le développement logiciel. Il travailla à l'élaboration de la méthode SCRUM avec Jeff Sutherland et Mike Beedle [Highsmith 2002a; ADM 1996a].

3.2.1.2 Évolution de ADM vers SCRUM

SCRUM est intimement liée à la firme ADM. Leur outil MATE, s'appuyait sur une grande connaissance des méthodologies. L'outil évolua de 1987 à 1993, pour devenir un outil de gestion de processus de travail collaboratif, orienté objet [ADM 1996b].

En 1994, AMD révisa son approche. Elle conclut que les processus utilisés contraignaient le développement et que ces mécanismes de gestion surchargeaient les gestionnaires et les développeurs. Un changement était nécessaire. L'entreprise a pris alors deux grandes orientations de recherche :

- Quel est le meilleur moyen de faire ressortir les connaissances tacites vers des connaissances explicites et publiées ?
- Quel processus de développement adopter pour produire des logiciels dans un environnement complexe, imprévisible et changeant du point de vue technique et d'affaire ?

Le modèle de processus qui pourrait rencontrer leurs objectifs, devait tolérer le chaos, être mesuré constamment et maximiser la communication. Le produit ne doit jamais être considéré complété, mais doit être le meilleur possible compte tenu des efforts qui auront été investis [Norton 2005].

3.2.1.3 Une nouvelle joute de développement

C'est Jeff Sutherland qui a entendu parler de SCRUM à partir du livre de DeGrace and HuLet Stahl de 1990 « Wicked Problems, Righteous Solutions», principalement suite à un article de Takeuchi et Nonaka dans leur publication du Harvard Business Review de 1986 « New New Product Development Game ». Sutherland les définit les auteurs comme les parrains du processus SCRUM, ils ont introduit cette notion à partir d'un simple paragraphe : *"The... 'relay race' approach to product development...may conflict with the goals of maximum speed and flexibility. Instead a holistic or 'rugby' approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements." [Takeuchi 1986]* Cet article présente une liste de caractéristiques pour travailler à la limite du chaos, tout en conservant le contrôle du projet. Le nom SCRUM fait référence à la mêlée que forment les joueurs de rugby lorsque le ballon est mis en jeu. Cette métaphore nomme les rencontres journalières que tient l'équipe

au cours du développement. Dans un environnement instable, les équipes sont appelées à être autonomes pour s'organiser. Le chevauchement des phases de développement favorise l'apprentissage multiple, le développeur améliore ses compétences d'architecte, de concepteur, de programmeur et de testeur. La fréquence des rencontres augmente le transfert des connaissances organisationnelles, le tout suivi par des contrôles subtils.

3.2.1.4 Les types de processus

Chez DuPont, Schwaber rencontra Babatunde Ogannaike qui rédigeait un document sur les contrôles des procédés industriels pour l'industrie chimique. Il dénota deux types de processus, ceux étant définis assez précisément pour être automatisés et les autres, moins définis, classés empiriques. Ces derniers ne pouvant être planifiés, des contrôles rapprochés s'imposaient [Highsmith 2002a]. En s'appuyant sur trois théories, Schwaber conclut que le processus de développement était empirique, plutôt que défini [Norton 2005].

> **Le principe d'incertitude en génie logiciel de Ziv,** qui statue que l'incertitude est inhérente et inévitable en développement logiciel.
>
> **Le principe d'incertitude des spécifications de Humphrey,** qui statue que pour les nouveaux logiciels, les spécifications ne seront pas complétées tant que l'utilisateur n'aura pas utilisé le logiciel.
>
> **Wegner's Lemma** statue qu'il est impossible de fournir toutes les spécifications pour un système interactif.

La figure suivante illustre les résultats d'analyse de chez ADM et le pourcentage de chance de réussite d'un projet de développement, en fonction de la complexité de son environnement. Les résultats démontrent qu'il y a moins de chance de terminer avec succès un projet complexe en suivant un processus défini plutôt qu'empirique.

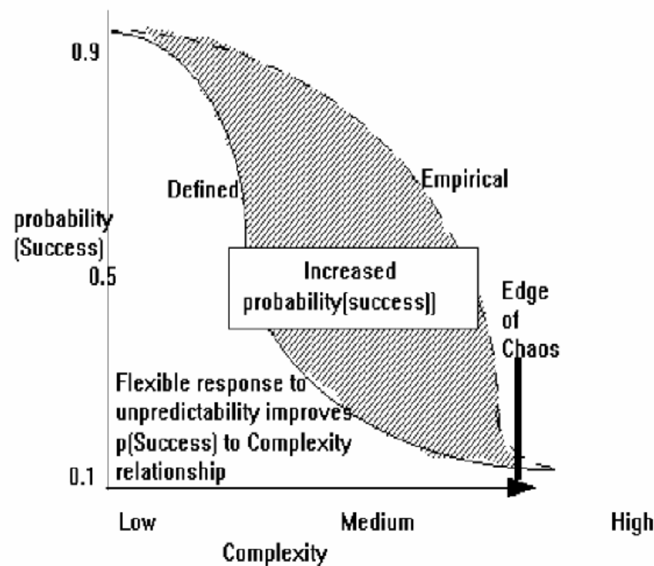


Figure 3-11 : Probabilité de succès en fonction de la complexité du projet

[Tiré de ADM 2003].

Un processus défini permet de prévoir les résultats en fonctions d'un contexte. Un tel processus peut produire un modèle théorique qui est automatisable. À l'opposé, si on ne connaît pas complètement un processus et que le seul moyen d'obtenir le résultat voulu est de le contrôler, ce processus est qualifié d'empirique. Les modèles des processus empiriques sont dérivés des variables observées et des contrôles implantés pour produire un résultat dans des limites prescrites. Nous devons le traiter comme une boîte noire [ADM 1996b; Rosenhead 1998]. Le travail d'un programmeur étant considéré de la sorte, il peut tout de même être contrôlé par des mécanismes de suivi à brève échéance. Moins un système est prévisible, plus le suivi doit être court. Les rencontres journalières contrôlent ainsi le processus.

3.2.1.5 L'amalgame de SCRUM

SCRUM s'inspire de sources diversifiées. Elle est une amélioration du processus itératif et de l'approche incrémentale pour le logiciel orienté objet, initialement documenté par Pittman, puis par Booch [Pittman 1993; Booch 1995]. Ce processus peut servir au personnel de gestion, mais d'une manière différente, tel que décrit par Graham [Graham 1994].

Plusieurs processus sont traités comme définis, alors que ce n'est pas le cas. Des résultats imprévisibles et incontrôlés en découlent. La première observation de petites équipes hautement performantes a été réalisée en 1986 par Takeuchi and Nonaka dans plusieurs firmes de haute technologie [Takeuchi 1986]. Une approche similaire appliquée au développement logiciel a été observée chez Borland par Coplien [Coplien 1994].

Après avoir été adaptée pour le développement en Smaltalk par Sutherland et pour Delphi par Schwaber, [Schwaber 1996] les analystes conclurent que SCRUM peut aussi être appropriée pour d'autres entreprises de développement [Aberdeen 1995].

3.2.2 Description de la méthode

3.2.2.1 Vocabulaire de SCRUM

Plusieurs éléments de SCRUM sont des métaphores référant au rugby. Le développement est perçu comme un jeu d'équipe. Au dire de Ken Schwaber « Les deux s'adaptent, sont rapides, s'organisent de manière autonome et ont peu de repos.» [Traduit et tiré de Highsmith 2002a]

Backlog produit : Liste des éléments techniques et fonctionnels à produire dans un avenir prévisible pour le produit complet. Inclue ce qui est défini et ce qui reste à préciser.

Sprint backlog : Liste des éléments techniques et fonctionnels à produire au cours d'un sprint. C'est un sous-ensemble du backlog produit. Elle comporte des éléments suffisamment définis pour qu'ils soient réalisés au cours de la période du sprint.

Sprint : Une période de 30 jours où un ensemble de travaux sont effectués pour créer un livrable.

SCRUM : Rencontre journalière de l'équipe où l'avancement et les empêchements sont révisés. Il s'agit du plus fréquent et plus bas niveau de contrôle.

Règlement des rencontres SCRUM : Protocole des réunions quotidiennes pour être efficaces. Afin de minimiser la logistique, une série de règles uniformise la dynamique des rencontres.

Équipe SCRUM : Équipe multidisciplinaire qui travaille sur les éléments du sprint. Toutes les personnes (vente, service clientèle, etc.) impliquées dans le projet sont invitées à se joindre à l'équipe.

3.2.2.2 Les trois phases de SCRUM:

Cette méthode se décline principalement en trois phases [Schwaber 1996].

La Phase initiale est consacrée à préparer le travail à faire. Cette démarche est linéaire et connue. Cette phase doit construire la liste des éléments que comportera cette version du produit. Elle comporte les objectifs suivants :

Planning -Architecture
Mise en place d'un backlog produit (liste des éléments à effectuer)
Définition de l'équipe
Analyse des risques – Budget

Les phases de sprints sont les phases itératives qui réalisent concrètement les éléments qui composent le produit. Ce travail étant plus chaotique, il est aussi qualifié de « boîte noire ». Elles comportent les objectifs suivants :

Un sprint backlog est sélectionné, cette liste sera suivie et réalisée.
Période d'environ 30 jours, au cours de laquelle l'équipe est isolée des influences extérieures. C'est-à-dire que la liste des éléments reste inchangée.
Réunion quotidienne SCRUM.
Comporte au dernier jour, une réunion post-sprint de présentation des résultats.

La Phase de clôture, elle aussi est linéaire et connue. Elle prépare la documentation, finalise les tests et rend la version fonctionnelle et présentable.

Documentation finale
Préparatif et livraison.

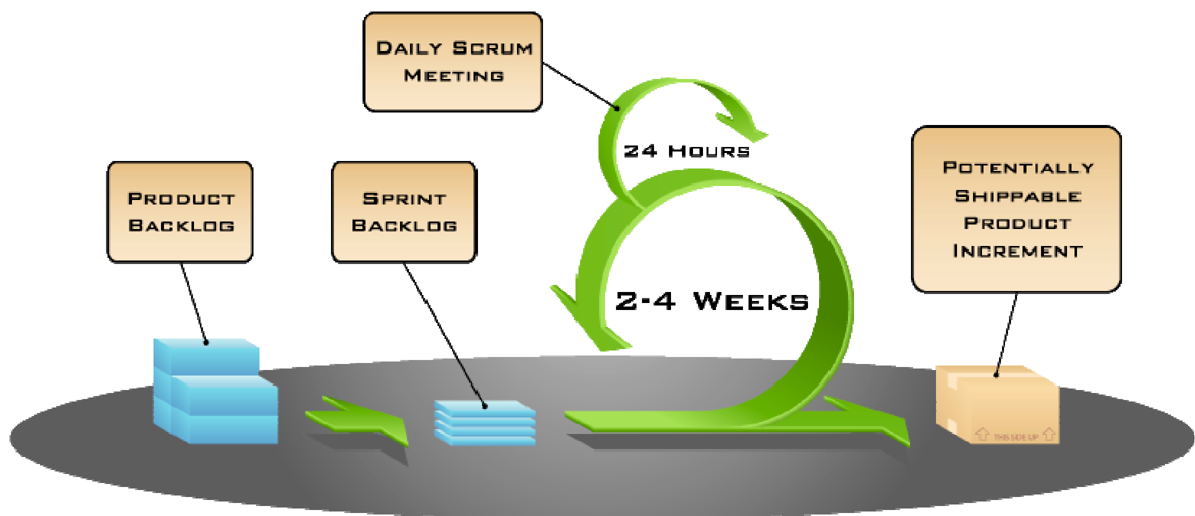


Figure 3-12 : Processus global de SCRUM

[Tiré de Cohn 2005a].

Phase initiale

SCRUM utilise une liste de caractéristiques envisageables qui composent le backlog du produit. Toutes personnes pouvant apporter quelque chose au projet sont invitées à participer, ce qui inclue les ventes, le marketing, le service à la clientèle, etc. Comme l'équipe technique participe avec le client à dresser cette liste, les caractéristiques incluent à la fois les fonctionnalités et les éléments techniques. Cette liste, d'ordre macroscopique, permet de dresser une architecture globale du produit.

La rencontre précédent un sprint, assure de choisir les éléments pour composer le backlog du sprint. Les caractéristiques sont choisies en fonction de leur valeur pour le client. En un premier temps, on identifie les caractéristiques. Ensuite on détermine les tâches pour réaliser ces caractéristiques. Une estimation plus précise peut alors être réalisée, afin de coordonner les efforts nécessaires et les ressources disponibles au cours du sprint. Cette liste sera réalisée au cours des 30 jours que durera le sprint.

Finalement, un but au sprint doit être déterminé selon une vision d'affaire. Cet objectif peut être atteint avec un sous-ensemble des caractéristiques ou de tâches demandées, afin de laisser un peu de latitude à l'équipe au cours du sprint. L'objectif étant la cible à atteindre afin que l'équipe arrive à concevoir ce qui doit être fait. Ceci conserve la vision de l'objectif et rappelle pourquoi quelque chose doit être fait. C'est ce qui permet une certaine latitude, afin d'adapter certains éléments pour atteindre l'objectif [Highsmith 2002a].

Sprint

Le sprint est la phase de réalisation itérative du produit, c'est le cœur de SCRUM. Il peut y avoir plusieurs sprints afin de créer une version livrable. La section suivante reprend en détail ces éléments.

Clôture

À la fin d'un des sprints de la version, le client peut décider que le produit est suffisamment complet en fonction de son échéancier, ses finances ou toute autre raison. Cette décision commande de préparer une version livrable. C'est à ce moment que le polissage du produit, l'implantation et la formation sont réalisés pour cette livraison.

Pour initier un sprint, il doit y avoir une liste d'éléments sélectionnés composant ainsi le backlog du sprint. Au cours de cette période d'environs 30 jours, l'équipe se concentre sur cette liste. Toutes les altérations (ajouts, retraits) seront relayées au backlog produit. L'équipe est isolée de l'influence extérieure pour cette période [BI 2001]. C'est au cours de cette période que l'équipe doit s'adapter à son environnement, qui est constitué des exigences, du temps, des ressources, des connaissances et de la technologie [VTT 2002].

Planification du sprint

Cette rencontre se divise en deux parties. La première vise à établir l'objectif des fonctionnalités du prochain sprint. La seconde, plus technique, vise à établir les moyens mis en places pour la réalisation de cet objectif.

Backlog du sprint

En fonction des éléments du backlog produit, les éléments prioritaires et pertinents avec l'objectif du prochain sprint sont sélectionnés. Lorsque la liste est établie, le sprint peut commencer. Cette liste reste stable jusqu'à la fin du sprint.

SCRUM journalier

Chaque jour, l'équipe se rencontre pour une période d'environs 30 minutes. Cette courte rencontre est appelée SCRUM et a pour but de mesurer la progression du sprint. C'est un des contrôles empiriques de SCRUM. Le chef de projet, appelé « SCRUM Master », demande à chaque participant de répondre à trois questions :

- Qu'est-ce que j'ai fait depuis le dernier SCRUM ?
- Qu'est-ce qu'il me reste à faire ?
- Comment ai-je l'intention de le faire ?

Le chef de projet a pour responsabilité de prendre les décisions que l'équipe n'est pas en mesure de prendre et d'éliminer les embûches rencontrées. L'objectif est à la fois d'avoir l'avancement du projet et de partager les connaissances entre les membres de l'équipe.

Rencontre de révision du sprint

Au dernier jour du sprint, se tient une rencontre informelle avec tous les intervenants afin de démontrer la progression. C'est à ce moment que la décision est prise pour poursuivre le projet vers un autre sprint ou conclure avec une livraison. Au terme de cette rencontre, il peut en découler de nouveaux éléments au backlog produit ou une nouvelle orientation.

3.2.2.3 Les contrôles

Les variables d'un projet de développement de système sont les risques, les fonctionnalités, les coûts, le temps et la qualité. Ces variables sont grossièrement estimées au début du projet. Chacune d'elle changera en cours de projet. Il existe une interrelation entre les variables. Par exemple, nous pouvons améliorer les fonctionnalités en investissant plus de temps et d'argent. Afin de permettre un suivi, différents contrôles existent dans SCRUM. Chaque élément peut être quantifié, afin de permettre l'extraction de métriques qui permettent le suivi [ADM 1996a]. On y retrouve notamment :

> Le **backlog projet** liste toutes les fonctionnalités et caractéristiques qui seront complétées dans l'ensemble du projet.
>
> Les **problèmes** sont des éléments qui doivent être solutionnés par un membre de l'équipe, afin de permettre la réalisation des objets liés au backlog produit.
>
> Un **changement** est l'activité réalisée pour résoudre un problème ou implanter une solution.
>
> Un **risque** peut être associé à un problème, un enjeu ou un élément du backlog.

Ces contrôles sont mesurés, corrélés, et suivis. Les principaux contrôles sont le backlog et les risques. Le backlog doit être de haut niveau pour commencer et raffiné en cours de projet. La liste des risques se rédige lorsque le backlog, les enjeux et les problèmes sont plus détaillés. Lorsque le logiciel est complété, les risques sont rendus à un niveau acceptable

L'annexe B (Observations sur SCRUM) discute des différentes possibilités et présente quelques extensions et solutions possibles concernant les problématiques fréquemment discutées. À la fin de cette annexe, vous trouverez une synthèse d'une seule page.

Conclusion sur SCRUM

Certains prétendent que SCRUM pourrait se voir étendre à divers secteurs de l'entreprise. C'est un retour aux sources, car à son origine, SCRUM ne s'adressait pas au développement logiciel [Sutherland 2005].

Puisant ses racines au milieu des années 1980, SCRUM amène une dynamique différente de la gestion de projets des développements logiciels. Elle propose un équilibre entre la portion définie et la portion chaotique du développement de produit. Cette méthode ne précise pas de technique de réalisation précise. Elle laisse cet aspect à la discrétion de l'équipe. Cette ouverture permet l'inclusion de pratiques provenant de d'autres méthodes.

En priorisant sur la valeur ajoutée par chaque composante du produit, SCRUM assure que ce dernier est toujours la meilleure valeur d'affaire possible pour l'investissement du client.

En détachant l'aspect logiciel et en portant cette approche à d'autre domaine d'ingénierie, SCRUM se permet de porter l'agilité aux autres secteurs de développement de produits.

3.3 Famille Crystal

3.3.1 Origine de Crystal

3.3.1.1 Son inventeur

Alistair Cockburn se présente comme un ethno-méthodologiste, il étudie la culture et les méthodologies de développement. Ayant beaucoup voyagé, il constate que la plupart des cultures fonctionnent. Bien que parfois étrange, ne pas comprendre l'aspect culturel peut mener à l'échec. Chaque pays, entreprise ou projet possède sa propre culture.

À travers ses expériences, il réalisa que la théorie ne s'appliquait pas partout et qu'une culture locale existe à chaque endroit. Il n'y a pas qu'une seule réponse à la question des méthodes. Il chercha à discerner les meilleures techniques. Il mit à l'essaie une première méthode et constata que les gens ne suivaient pas ce qui était prescrit. Il comprit qu'il y avait quelque chose de plus complexe : la nature humaine.

Pour lui, une méthodologie se définie par un ensemble d'éléments de communication et de politique, afin de coordonner les efforts d'une équipe. Cet ensemble de conventions défini la méthode. Ce que fait un groupe de personnes pour se coordonner est une méthodologie[6]. Cockburn est aussi motivé par l'introspection créatrice nécessaire à la conception logiciel.

Il est un des rares méthodologiste à creuser le passé, afin d'analyser ce qui fonctionne efficacement. Il découvrit plusieurs choses intéressantes. Premièrement, pratiquement personne n'applique ce que l'on retrouve dans les livres d'experts. Peu de gens disent qu'ils ont essayé et ceux qu'ils le disent, ne le recommande pas. Deuxièmement, quels que soient les outils ou les technologies utilisés, aucun ne prédit la performance. Le seul point déterminant, est que l'équipe collabore bien [Highsmith 2002a].

3.3.1.2 La famille de méthode Crystal

Crystal est une plate-forme méthodologique. La famille de méthode Crystal comporte présentement quatre membres, qui ne sont pas encore tous documentés. Présentement, seul Crystal Orange et Clear ont été décrit dans différents ouvrages [Cockburn 1997; Cockburn 2004]. L'analogie avec les couleurs de Crystal, réfère à son opacité, plus la méthode est complexe plus sa couleur est foncée. Pour qu'une équipe puisse choisir la méthode qui lui convient, on présente souvent cette famille sous forme de matrice.

L'axe des X représente la grosseur de l'équipe. Le nombre de personne impliquée sur un projet, détermine les éléments de communication qui devront être utilisé. La qualité de la communication est fondamentale pour Crystal. Plus il y a de personnes impliquées, plus les moyens de communications sont formalisés.

L'axe des Y représente les dommages potentiels pouvant avoir un impact sur la qualité du produit. Le niveau critique du système s'étend de la perte de confort, la perte d'argent discrétionnaire, la perte d'argent essentielle ou la perte de vie. Plus le produit comporte un impact lourd, plus il y aura de pratiques d'assurance qualité du produit.

---

[6] La méthodologie, est une méta méthode, méthode des méthodes ou classe de méthodes, une sorte de boîte à outils, dont chaque outil est une méthode singulière appropriée à résoudre une énigme en particulier. [Wikipedia 2005] Méthode : http://fr.wikipedia.org/wiki/M%C3%A9thodologie

Life (L)	L6	L20	L40	L80
Essential Money (E)	E6	E20	E40	E80
Discretionary Money (D)	D6	D20	D40	D80
Comfort (C)	C6	C20	C40	C80
	Clear	Yellow	Orange	Red

Figure 3-13 : Matrice de la famille Crystal

[Tiré de Cohn 2004]

3.3.2 Description de la méthode

Crystal Clear est une méthode ouverte qui permet d'intégrer des pratiques provenant de d'autres méthodes. Ce qui constitue le noyau de cette méthode, c'est les propriétés appliquées à travers différentes stratégies et techniques [Cockburn 2004].

3.3.2.1 Les sept propriétés de Crystal Clear

Propriété 1. Livraison fréquentes

Livrer fréquemment comporte plusieurs avantages. Cela démontre le progrès et permet aux utilisateurs de valider la conformité du produit. Ce retour d'expérience guide l'équipe de développement et l'encourage à continuer. Cette propriété permet aussi de raffiner la méthode de déploiement. Une livraison mensuelle est conseillée. Une équipe adoptant cette propriété peut se demander: "Au cours des six derniers mois, avez-vous livré au moins deux fois le code testé et utile à vos utilisateurs?"

Propriété 2. Réflexion d'amélioration

En cours de développement, l'équipe est appelée à s'analyser pour faire ressortir les pratiques efficaces et celles qui doivent être améliorées ou abandonnées. L'équipe adapte sa méthode. Elle ajuste ses conventions, afin de trouver son point d'équilibre. Une équipe

adoptant cette propriété peut se demander: « Au cours des deux derniers mois, votre équipe s'est-elle réunie pour discuter de la manière dont elle travaille? Savez-vous ce qui vous aide, vous ralenti et ce qui peut être amélioré? »

Propriété 3. Communication osmotique

La communication est le point fort de Crystal Clear. Cette méthode demande que l'équipe soit très proche et dans une aire ouverte. Il faut que les gens soit « à porter d'oreille » afin que les conversations d'arrière plan puissent être comprises. C'est une manière très rapide et efficace pour obtenir les commentaires des autres membres de l'équipe. Lorsque quelqu'un pose une question, tout le monde peut y répondre ou corriger la réponse de celui qui l'a fourni. Une équipe qui adopte cette propriété peut se demander: « Est-ce que cela prend moins de trente secondes afin de poser une question, face à face, à la personne qui connaît la réponse? Percevons-nous régulièrement des éléments pertinents à travers les conversations des autres? »

Propriété 4. Sécurité personnelle

Les membres de l'équipe doivent pouvoir discuter librement des choses qui les dérangent, sans avoir peur des représailles. C'est une preuve de confiance et cela signifie que la qualité des échanges est de nature transparente. Il faut que les membres de l'équipe puissent admettre leur ignorance, leurs erreurs ou leur incapacité à accomplir une tâche. Lorsqu'il n'y a pas de crainte à cet égard, les gens échangent les informations plus librement. Cette propriété s'appuie sur des études qui prouvent une corrélation entre la confiance et la performance d'une équipe. Une équipe adoptant cette propriété peut se demander: « Puis-je dire à mon supérieur que j'ai sous-estimé le projet de 50%? Pouvons-nous débattre sur la conception des autres et manifester un désaccord amical? »

Propriété 5. Concentration dédiée

Chaque personne doit savoir ce qu'il a à faire et avoir le temps requis pour s'y consacrer. Connaître l'objectif aide à motiver le développeur qui comprend mieux l'importance de sa tâche. Pour se motiver, une personne doit pouvoir investir suffisamment de temps pour démontrer une progression. Il est possible que des plages de temps sans dérangement soient

requises, afin d'atteindre cet objectif. Par exemple, certaines équipes prohibent les réunions en matinée. Une équipe adoptant cette propriété peut se demander: « Est-ce que tout le monde connaît les deux éléments prioritaires sur lesquels ils doivent travailler? Est-ce-je peux travailler sur mes priorités, au moins deux jours de suite par semaine ou deux heures de suite par jour? »

Propriété 6. Accès facile aux utilisateurs experts

Contacter facilement l'utilisateur comporte plusieurs avantages. Cela permet de déployer et tester les livraisons. L'accès facile permet de fournir du feedback rapide sur la qualité du travail et sur les décisions de conception à prendre, tout en fournissant des spécifications à jours. Il existe différentes méthodes pour faciliter cet accès. La première, est de rencontrer périodiquement l'utilisateur. La seconde, consiste à intégrer l'utilisateur à l'équipe de développement. Finalement, un développeur peut être formé auprès des utilisateurs, afin de connaître les différentes tâches supportées par le système à produire. Une équipe adoptant cette propriété peut se demander: « Pouvons-nous obtenir les réponses des utilisateurs en moins de trois jours? Le pouvons-nous en quelques heures? »

Propriété 7. Environnement technique favorable

Certains éléments techniques favorisent l'agilité. Les outils utilisés permettent une souplesse dans la manipulation du code. Les meilleures équipes utilisent conjointement les trois éléments :

- **L'automatisation des tests** permet de valider rapidement les derniers ajouts faits au code. Utilisé largement pour les tests unitaires, il est aussi possible d'étendre cette couverture vers les tests d'acceptation ou d'interface utilisateur.
- **La gestion de la configuration** permet de travailler sur différentes parties du code, sans se soucier « d'écraser » un travail en cours de réalisation. Elle permet de figer la configuration d'une livraison et de poursuivre l'évolution du produit.
- **L'intégration fréquente** permet aux équipes de valider la compatibilité de leurs améliorations avec l'ensemble du système. Plus l'intégration est régulière, plus le délai de détection des erreurs est rapide. Conjointement avec l'automatisation des tests, la détection d'erreurs est extrêmement efficace.

Une équipe adoptant cette propriété peut se demander: « Pouvons-nous lancer les tests sans que personne ne soit présent? Est-ce que tous les développeurs vérifient leur code dans

l'outil de gestion de configuration? Est-ce que le système est intégré au moins deux fois par semaine? »

3.3.2.2 Les stratégies et les techniques

Crystal Clear n'impose pas de stratégie ou de technique précise. Elle en propose certaines, afin d'aider le démarrage. L'équipe déterminera en cours de projet ce qui est le plus avantageux pour elle. Crystal comporte une révision périodique sur la manière de travailler, ce qui permet à l'équipe de s'ajuster [Cockburn 2004].

Les stratégies

Exploration 360°

Au début du projet, l'équipe est invitée à faire un tour d'horizon du projet. Cette stratégie permet au départ de fixer quelle est la valeur d'affaire et ses spécifications. Cette étape de haut niveau permet de planifier le projet, les technologies, de composer l'équipe, de définir la méthode et les conventions de travail utilisés. Cette étape de planification peut prendre quelques jours à une semaine. À la fin de cette étape, les décideurs sont en mesure de choisir si le projet est une bonne opportunité ou s'il est préférable de ne pas procéder.

Victoire rapide

Gagner est une force qui aide les équipes à avoir confiance en elle-même. Les petites victoires renforcent le groupe et favorisent l'esprit d'équipe. Si cette force surgit en début de projet, elle peut permettre d'accélérer la cohésion de l'équipe dès le départ. Certaines méthodes proposent de commencer par le plus complexe en premier, afin de valider que les difficultés latentes soient surmontées. Paradoxalement, l'équipe doit s'adapter et se faire confiance, ils ont donc deux difficultés importantes à confronter simultanément. La stratégie que propose Crystal est de prendre un élément facilement réalisable, afin de valider la manière de travailler avant d'attaquer les problèmes les plus complexes. De cette manière, un projet techniquement réalisable ne sera pas abandonné pour des raisons d'organisation du travail.

Squelette marchant

Cette stratégie complète aisément celle de la victoire rapide. Son objectif est de concevoir une architecture minimale pour que le produit soit fonctionnel. Il s'agit parfois d'une fonction primaire du système tel que de permettre une inscription dans une base de données. Ceci permet de baliser la démarche technique qui sera suivie. Elle valide le fonctionnement des couches nécessaire au système. Le code réalisé à ce niveau est permanent. Il ne s'agit pas d'un prototype ou une preuve de faisabilité technique. C'est l'ossature sur laquelle est incrémentée les autres fonctions. Cette stratégie est aussi utilisée dans d'autres méthodes [Poppendieck 2003a].

Ré architecture incrémentale

L'architecture doit évoluer à partir de son origine pour répondre aux contraintes d'affaires et technologiques. Révisée, l'architecture est une des propriétés fondamentales de l'agilité logiciel. Il est possible que certains architectes très expérimentés puissent concevoir une architecture ayant une longue portée. Par contre, elle ne devrait pas excéder la vision de l'architecte et les contraintes établies. Une architecture minimaliste et simple s'avère facilement évolutive. Les fonctionnalités sont disponibles plus rapidement, ce qui accélère la réception des commentaires des utilisateurs.

Radiateurs d'information

Les radiateurs d'informations sont des outils d'affichage, illustrant la progression du projet. Souvent sous forme d'affiches, ils doivent être installés à la vue et comporter des informations facilement compréhensibles. Ces outils permettent d'informer tous les passants, ce qui réduit les dérangements causés par les questions d'avancement. L'information couramment affichée, couvre les fonctionnalités en cours de développement, les assignations ainsi que différentes métriques, tel que le nombre de cas de tests développés ou le constat des réunions.

Les techniques

Façonnement de la méthode

Cette technique se décompose en deux étapes. Elle débute par une série d'entrevue, afin d'élaborer les forces et les faiblesses de l'équipe. Il en résulte une liste d'expertises, des bonnes et des mauvaises pratiques expérimentées. La seconde étape reprend cette liste, afin de composer une méthode. L'équipe est appelée à choisir les éléments qui composeront la méthode employée en cours de projet. Cette série de conventions sert de point de départ au projet.

Atelier de réflexion

Les ateliers de réflexion sont complémentaires au façonnement de la méthode. Après chaque livraison, il est conseillé à l'équipe de se consulter pour analyser leur méthode de travail. Elle discute de ce qui fonctionne bien et de ce qui devrait être corrigé ou abandonné. Afin d'aider la réflexion, il est suggéré de dresser un tableau séparé en trois sections dans lesquels sont listés les différents éléments de la méthode. Une section est consacrée à ce qui doit être conservé. Une autre section est consacrée à ce qui ne fonctionne pas bien et ce qui cause des problèmes. Finalement une section est consacrée à lister ce qui devrait être essayé.

Blitz de planification

C'est l'occasion pour les décideurs et les utilisateurs experts de contribuer au projet avec l'équipe de développement, afin de produire le plan et l'échéancier du projet. Un peut comme XP [Beck 2000], les participants indiquent sur des cartes les différentes tâches à réaliser en cours de projet. On indique sur la carte une brève description de la tâche, la personne en charge et une estimation sommaire. Ces tâches sont ensuite groupées, formant ainsi les itérations. L'ordre des itérations représente la priorité, l'effort total et individuel qui sera demandé au cours de sa réalisation.

Estimation Delphi

Cette technique d'estimation a pour objectif de prévoir l'effort global du projet. Elle s'inspire de la technique « Wideband Delphi Technique » développé par la défense américaine [Wikipedia 2006h]. Son objectif est d'estimer les coûts, la durée, les ressources et de planifier les itérations. Cette technique requiert de réunir une équipe d'experts qui doit estimer différents aspects qui composeront le logiciel. Lors de la première phase, chaque expert doit faire sa propre estimation, qui sera discutée en groupe par la suite. La seconde étape consiste à planifier qui sera affecté aux différents éléments prévus. Cette seconde phase est importante, afin de s'assurer d'avoir les personnes requises au bon moment. La somme des efforts formera l'estimation initiale. La planification des itérations met ensemble les cas d'utilisation similaire, pour composer les différents livrables.

Rencontre journalière

De courtes rencontres journalières, d'environs 15 minutes sont suggérées. Cette technique ne ce cache pas de provenir de SCRUM *(Voir SCRUM journalier).* Le suivi journalier demeure la meilleure manière de suivre les travaux et de partager l'information.

Conception interactive

La conception centrée sur l'utilisation proposée dans Crystal Clear, provient de Jeff Patton [Patton 2002]. Elle demande la participation du client, afin de modéliser le processus de travail que devra supporter le système. Cette approche est centrée sur l'utilisation et permet de concevoir les interactions et les interfaces utilisateurs. De cette manière, le décideur, les utilisateurs et les développeurs partagent une compréhension commune du système. Une fois la conception complétée, il est simple de classifier les fonctionnalités par importance pour l'organisation. Cette technique préconise l'utilisation de papier « post-it » et de crayons, afin que tous les participants soient à l'aise d'intervenir dans la conception.

Processus miniature

Comme tout nouveau processus dans l'organisation, il est bon de le mettre à l'épreuve à petite échelle, afin de valider son fonctionnement. Ce que l'on appelle le processus miniature, consiste à prendre quelques heures afin de simuler son fonctionnement. Les oublis et les erreurs sont alors découverts rapidement. De cette manière, tous les gens impliqués ont une meilleure compréhension de leur rôle et de la place qu'ils occupent dans ce processus. Suite à cette expérimentation, les gens adhèreront plus facilement au processus.

Programmation côte à côte

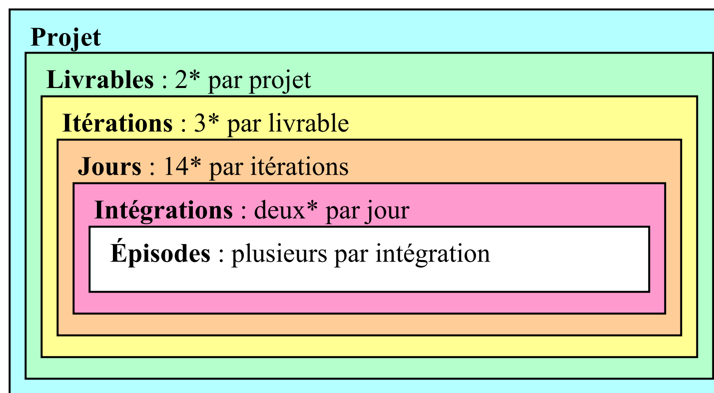
La programmation en pairs de XP, consiste à coupler deux développeurs assignés à la même tâche, sur un seul poste de travail. La programmation « côte à côte » de Crystal, n'impose pas de travailler à deux sur une même assignation. Il suggère de mettre les deux développeurs assez proche l'un de l'autre, pour leur permettre de voir l'écran de l'autre sans effort. Cette technique est une amplification de la communication osmotique. Il a été prouvé que les revues de conceptions et de codes sont économiques et comportent moins de défauts.

Tableau de progression

Le tableau de progression est un outil de communication qui illustre la planification et la progression du projet. Il se construit à travers le temps pour afficher la progression du projet. Il peut être présenté de manière traditionnelle, marquant une progression cumulant les efforts réalisés à travers le temps. Moins connu, le tableau régressif considère que 100% du travail reste à faire et que l'objectif à atteindre est 0. Il comporte un effet positif sur le moral de l'équipe qui voit le travail à faire, diminuer. Ce tableau est d'autant plus efficace si les repères sont assez rapprochés et fixes. Les itérations pouvant servir de repères, les dépassements sont constatés rapidement et peuvent être mieux contrôlés.

3.3.2.3 Cycle de développement

Comme les autres approches agiles, le modèle itératif incrémentale est repris. Crystal Clear est composé de cycles imbriqués. La méthode proposée nous permet de schématiser les cycles et leurs répétitions comme une série d'imbrications.



*valeur à titre indicatif, l'équipe doit déterminer elle-même cette valeur.

Figure 3-14 : Imbrication des différents cycles de Crystal.

[Inspiré de Cockburn 2004]

Le projet

Le projet est l'unité de base et peut être de durée variable. Il comporte une étape de planification initiale, au moins deux cycles de livraisons et une étape de fermeture.

La planification initiale vise à recruter l'équipe principale en charge du projet. C'est à ce moment que l'équipe fait l'exploration 360°, qui peut résulter par l'abandon du projet. On y détermine les conventions de la méthodologie qui seront suivi. Finalement, on y construit une planification initiale du projet. Cette planification de haut niveau comporte un aperçu des différents livrables à venir. Ensuite, commence les cycles de livraisons.

Lorsque tous les cycles de livraisons sont complétés, l'équipe procède à la clôture du projet, par une dernière réflexion sur son déroulement afin d'en dresser le bilan formateur.

Les livraisons

Le cycle de livraison peut durer d'une semaine à trois mois. Il se termine par la mise en place d'une version du produit fonctionnel pour l'utilisateur. Un cycle de livraison comporte une révision de la planification, une ou plusieurs itérations, une étape de livraison et une séance de réflexion.

La révision de la planification est utile à partir de la seconde livraison. L'équipe est en mesure de mieux évaluer le travail et peut réviser ce qui est prévu pour la livraison qui débute. Cette opération se fait conjointement avec le client, afin de tirer le maximum de valeur de la livraison. Il peut en résulter de remplacer l'équipe, changer la portée ou l'échéancier du projet ou trouver une solution plus créatrice pour arriver dans les délais avec l'équipe en place.

Suite aux itérations, arrive l'étape de livraison à l'utilisateur. C'est-à-dire, un produit fonctionnel et utile pour une véritable production. Étant donné les efforts de formation et de documentation que peuvent représenter une livraison, une durée de trois ou quatre mois est suggérée. Complémentairement, il est possible de déployer des versions intermédiaires à des petits groupes d'utilisateurs, afin d'obtenir le feedback nécessaire à la progression du projet.

À la fin de ce cycle, une séance de réflexion demande que l'on révise deux aspects du projet. D'une part, une réflexion sur le produit en cours de réalisation et d'autre part une réflexion sur les conventions mises en place qui composent la méthode.

Les itérations

Une itération peut durer d'une semaine à trois mois. Ce cycle prend en charge une portion des travaux d'une livraison. Elle comporte une étape de planification et un nombre variable de jours de travail. Elle se termine par une séance de réflexion et une célébration.

Pour une itération, la planification affecte les tâches aux membres de l'équipe. Les développeurs sélectionnent les tâches qu'ils estiment pouvoir réaliser au cours de l'itération. Lorsque chacun sait ce qu'il a à faire, le travail journalier débute.

À la fin de l'itération, les membres de l'équipe participent à une courte séance de réflexion et de célébration. L'objectif de cette activité est surtout de marquer un temps d'arrêt, afin de prévenir l'épuisement. Cette pause réconfortante évite d'épuiser les gens.

Les journées

Chaque journée comprise dans une itération comporte une rencontre de l'équipe de développement et aucune ou plusieurs intégrations. Les rencontres journalières effectuent le contrôle et le suivi des réalisations. Consulter la section sur les techniques pour plus de détails.

Les intégrations

Une intégration comporte plusieurs épisodes, se termine par une compilation et des tests. Dépendamment du projet, une intégration peut se produire à toutes les demi-heures ou aux quelques jours. Un environnement technique bien adapté, permet d'accélérer sa réalisation. Une série de tests sont réalisés à chaque intégration, afin de valider immédiatement le bon fonctionnement du produit avec les nouveaux ajouts.

L'épisode

Un épisode est la portion qui inclut le design, le codage et la vérification. Dans un contexte agile, il est bon de spécifier que le design et le codage sont des activités conjointes. Le développeur est libre de choisir le design qu'il juge adéquat, afin de réaliser sa tâche. Il doit aussi tester son code avec des tests unitaires automatisés de préférence.

L'annexe C (Observations sur Crystal Clear) présente une analyse, comporte quelques comparatifs et compléments d'information sur cette méthode. À la fin de cette annexe, une grille de métriques agiles permet de vérifier le niveau d'agilité d'une équipe.

3.3.3 Conclusion sur Crystal Clear

Cockburn attache une attention particulière à la culture. Ayant travaillé dans divers pays, il conclut que toutes les équipes peuvent atteindre leurs objectifs. Le développement logiciel est un jeu de coopération et de communication.

Crystal fait ressortir les principes communs qui régissent les équipes gagnantes. Une méthode n'est qu'une série de conventions, qu'un groupe d'individu suivent. L'équipe est invitée à construire sa propre méthode sur la plate-forme que propose Crystal. La méthode qui en résulte recherche l'efficience, doit être viable pour les membres de l'équipe et amène le projet dans une zone sécuritaire améliorant ainsi ses chances de succès.

3.4 Autres méthodes agiles

Pour qu'une méthode soit considérée agile, elle doit satisfaire aux critères du manifeste [AA 2005a]. Dans ce cadre, une série de méthodes documentées peuvent être élaborée. Dans cette section, nous présentons les principales méthodes de l'alliance agile, qui se qualifient face au manifeste et qui se retrouvent régulièrement dans les ouvrages de synthèse [Highsmith 2002a; BI 2001; VTT 2002]. L'objectif de ce mémoire n'est pas d'être exhaustif et de décrire chacune d'elle, cette section propose plutôt une brève description de celle-ci. L'annexe D (Autres approches complémentaires) survole aussi d'autres approches liées aux valeurs agiles.

3.4.1 Lean Software Development

Introduit par Robert N Charrette, Lean Development (LD) est la plus stratégique des approches agiles. Elle s'inspire du « Lean Manufacturing » de Deming [Deming 1982] qui révolutionna l'industrie automobile dans les années 1980, aussi connu sous le nom de « l'approche Toyota ». Elle propose de réaliser un produit équivalent à une approche traditionnelle CMM niveau 3, en un tiers du temps, pour un tiers des coûts et comportant un tiers des défauts [Highsmith 2002a].

« Lean development se définit comme une approche systématique qui élimine le gaspillage à travers l'amélioration continue et le perfectionnement du produit impliquant la participation du client. » [Traduit et tiré de Kilpatrick 2003]

Bien qu'à l'origine de cette approche, Charrette ait peu écrit sur LD [Norton 2005], c'est en 2003 que Tom et Mary Poppendieck présentèrent leur adaptation intitulée : Lean Software Development (LSD) [Poppendieck 2003a]. Ce livre comporte une série d'outils pour supporter cette approche.

« Lean Software Development, réduit les défectuosités et les délais de livraison qui se régularisent et qui augmentent la valeur d'affaire » [Tiré et traduit de Windholtz 2006].

Les domaines de gaspillage peuvent se transférer au logiciel. Ce qui nous donne le tableau suivant.

Domaine de gaspillage original	**Domaine transféré au logiciel**
Surproduction	Fonctionnalités superflues
Inventaire	Spécifications
Étapes superflues du processus	Étapes superflues
Déplacements	Trouver l'information
Défectuosités	« Bugs » non détectés aux tests
Attentes	Attentes de décision (incluant le client)
Transport	Transferts manuels

Tableau 3-8 : Domaines de gaspillage appliqués au logiciel

[Traduit et tiré de Kumar 2005]

Lean software n'est pas une méthodologie. C'est une manière de penser, basée sur sept principes [AA 2005c; Poppendieck 2003b] :

- **Éliminer le gaspillage** – Si le client ne valorise pas un élément ou si cela ralenti la vitesse de livraison au client, c'est du gaspillage. Ne le faites pas.
- **Augmenter l'apprentissage** – Le développement est une activité de découverte et de feedback. Livrez par petits morceaux pour réduire l'incertitude et permettre au client de réagir.
- **Décider le plus tard possible** – Retarder les prises de décisions. Conservez différentes options ouvertes le plus longtemps possible, afin d'avoir la meilleure information possible lors de la prise de décision.
- **Livrer le plus rapidement possible** – La meilleure mesure de maturité d'une organisation est sa cadence et sa régularité à livrer de la valeur ajoutée.
- **Renforcir l'équipe** – L'équipe doit définir son processus. Fournir la formation et le leadership qu'ils ont besoin.
- **Construction intègre** – L'ingénierie simultanée augmente la qualité de la communication, élément essentiel pour l'intégrité du système.
- **Vision globale** – La décomposition amène des optimisations partielles. Concentrez-vous sur l'ensemble du résultat.

Bien que cette approche ne préconise aucune pratique précise. L'ensemble de ces principes rejoint les valeurs du manifeste agile, ce qui lui permet de se qualifier comme approche agile.

3.4.2 Dynamic System Development Method

Dynamic System Development Method (DSDM) ou Méthode Dynamique de Développement de Système, s'inscrit comme une formalisation du RAD des années 1980. Elle a été développée en Grande Bretagne par un consortium de sociétés en 1994 [Fowler 2005]. Il s'agit en fait d'un canevas plutôt que d'une méthode. Son évolution a même porté un ajustement sur l'acronyme de DSDM par Dynamic Solution Delivery Model ou Modèle Dynamique de Livraison de Solutions [Highsmith 2002a]. Elle priorise de donner à l'entreprise ce qu'elle a besoin, quand elle en a besoin. Elle assume fondamentalement que rien n'est parfait du premier coup, mais que 80% des fonctions utiles peuvent se réaliser en 20% du temps [DSDM 2005].

Cette approche inverse la pyramide de planification. L'approche traditionnelle fixe l'ampleur du projet et amène une variation sur le temps et les ressources. Inversement, DSDM fixe le temps et les ressources en laissant l'ampleur variable. [VTT 2002].

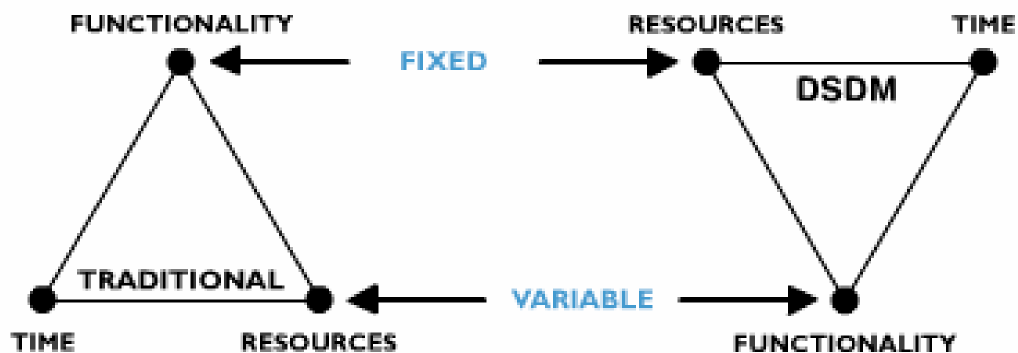


Figure 3-15 : Pyramide inversée de DSDM

[Tiré de DSDM 2005]

Neuf principes définissent le DSDM. Tous ces principes s'accordent avec le manifeste agile, c'est pourquoi elle se qualifie de méthode agile [DSDM 2005].

- L'implication des utilisateurs est obligatoire.
- L'équipe est autorisée à prendre des décisions.
- Livrer fréquemment le produit.
- Le principal critère d'acceptation est l'adaptation au besoin d'affaire.
- Le développement itératif et incrémental engendre une solution d'affaire appropriée.
- Toutes modifications en cours de développement sont réversibles.
- Les besoins sont établis sous forme de synthèse.
- Les tests sont intégrés pendant tout le cycle de vie.
- La collaboration et la coopération sont essentielles.

Le cycle de développement de DSDM propose cinq étapes, les deux premières sont séquentielles et les autres font partie des cycles itératifs. Le modèle proposé s'applique à tous les domaines qui touchent le projet. Cette méthode s'appuie beaucoup sur la construction de prototypes, à cause de ses origines. Par contre, cette notion doit être clarifiée, car les prototypes ne sont pas jetables. Le code qui les compose est complété dans la suite du développement [Highsmith 2002a].

3.4.2.1 Cycle de développement de DSDM

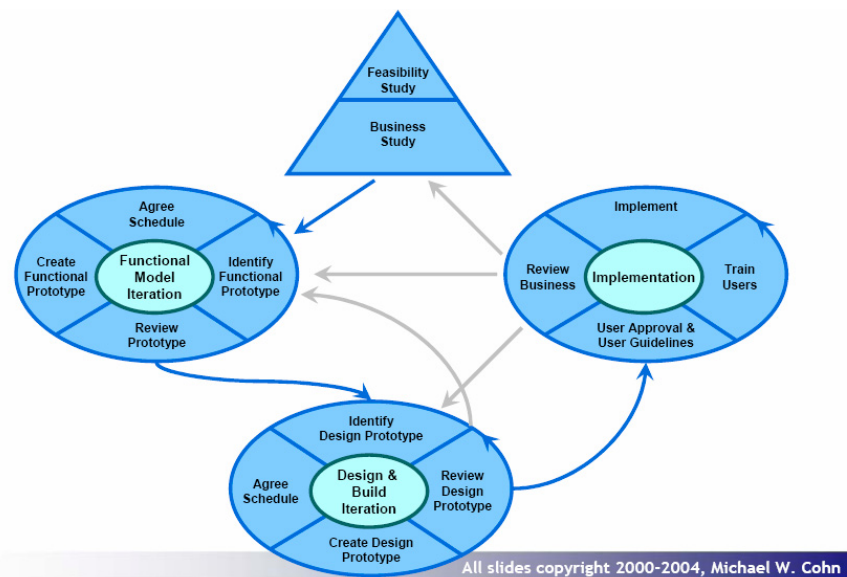


Figure 3-16: Cycle de DSDM

[Tiré de Cohn 2004].

L'étude de faisabilité a pour objectif de proposer une solution au problème posé. Elle estime les coûts et la faisabilité technique, elle peut même présenter un prototype. Cette étape décrit brièvement la solution proposée et indique si l'utilisation de DSDM est favorable pour mener le projet à terme.

L'étude du fonctionnement d'affaire couvre l'aspect du processus d'affaire à automatiser et requiert la participation des utilisateurs. L'étude permet de prioriser les besoins et de dresser l'architecture de haut niveau du système.

Le modèle fonctionnel itératif a pour objectif de préciser le besoin sur l'aspect fonctionnel des traitements et données. Cette étape produit des modèles d'analyse et aussi des prototypes qui intègrent une grande partie des spécifications qui ne sont pas décrits textuellement. Comme plusieurs méthodes agiles, DSDM comporte peu de documentation, mais ses différents prototypes compensent cet aspect.

La conception et réalisation itérative sont conditionnelles à l'accord d'un prototype. Elles raffinent le prototype et réalisent le module conformément au standard. Il en résulte un produit utilisable et testé. Le cycle fonctionnel et celui de la conception sont généralement abordés, subséquemment par domaine d'affaire.

La mise en œuvre comporte la migration et l'implantation en production du système. Cette phase couvre la formation des utilisateurs et la rédaction de la documentation.

Un aspect intéressant de DSDM, est la classification des besoins. Comme le projet laisse l'ampleur variable, une classification des besoins est utile. Cette classification suit la règle de MoSCoW. Cette règle fait référence à l'acronyme anglais : Must, Should, Could, Won't et qui peut se traduire par un ordre d'importance :

- Doit fondamentalement être présent pour le succès du projet.
- Serait important, mais ne compromet pas le projet.
- Pourrait être utile, mais ne comporte aucun impact.
- Pas pour l'instant, pourra être ajouté par la suite.

Le consortium améliore, promeut, forme et certifie les entreprises sur DSDM. Présentement à sa version 4.2, son utilisation se fait sous licence. Cette méthode est conforme avec ISO 9001. Il est sous-entendu que se faire certifier par DSDM, facilite la certification ISO.

3.4.3 Feature Driven Development

Feature Driven Development (FDD), ou en français : Développement Dirigé par les Fonctionnalités, a été développé par Peter Coad, Jeff de Luca et Stephen Palmer. Cette méthode priorise la gestion des risques, en s'appuyant sur de courtes itérations pour donner des fonctionnalités tangibles à l'utilisateur. Celui-ci est rapidement informé de l'avancement, ce qui diminue les risques. Selon ses créateurs, celle-ci peut-être utilisée dans le développement de systèmes critiques [VTT 2002].

Elle ne prescrit pas de pratique de programmation précise, mais certains rôles sont définis [BI 2001]. FDD est plus un guide qu'un processus prescrit. Selon ses inventeurs, FDD est capable de s'adapter à de gros projets [Highsmith 2002a].

Cette méthode comporte 5 grandes étapes, dont les deux dernières se trouvent dans le cycle itératif [Fowler 2005].

- Développer un modèle général.
- Construire une liste de fonctionnalités.
- Planifier par fonctionnalités.
- Designer par fonctionnalités.
- Construire par fonctionnalités.

Les étapes de départ sont révisées en cours de projet avec beaucoup moins d'ampleur. Le graphique suivant illustre les différentes étapes et le pourcentage approximatif d'effort requis au départ et par la suite dans le projet.

3.4.3.1 Cycle de développement de FDD

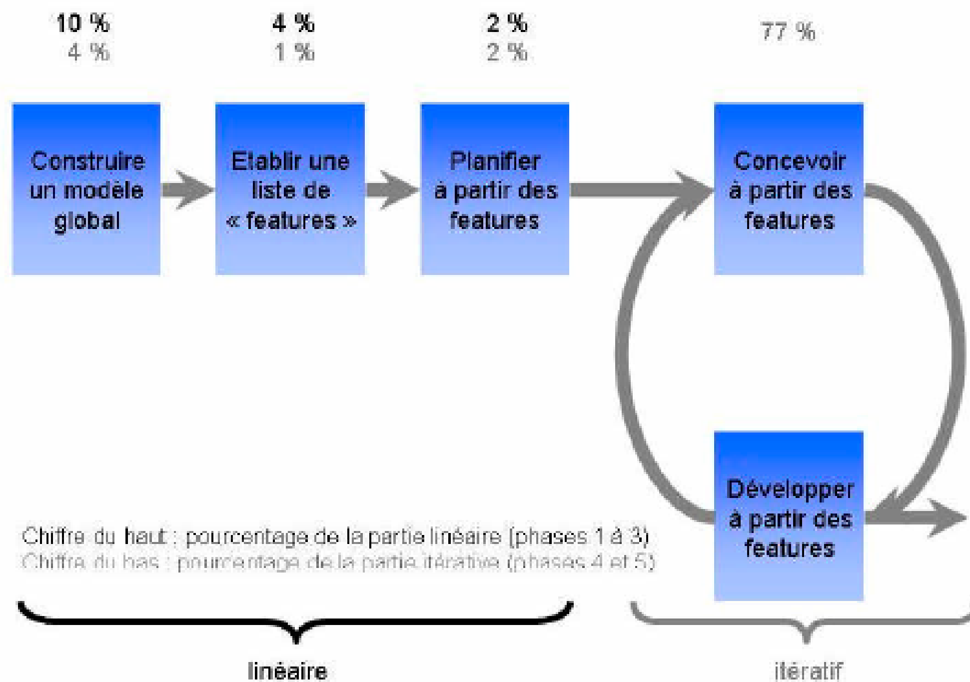


Figure 3-17: Cycle de développement de FDD

[Tiré de BI 2002].

La première étape est de concevoir le **modèle général** du système. Cette étape permet de délimiter le système et son contexte. Les cas d'utilisations de haut niveau sont aussi conçus à cette étape.

La seconde étape consiste à établir la **liste complète des fonctionnalités**. Avec la participation du client, cette étape met en évidence les fonctionnalités valables pour lui. Les caractéristiques définies sont complètement orthogonales avec le modèle objet. Il faut que le client conserve une compréhension du système [Highsmith 2002a]. Une notion intéressante de cette approche, est que les fonctionnalités sont décrites selon une syntaxe précise, qui standardise leurs énoncés.

> Fonctionnalité = <action> *LE* <résultat> *PAR* | *POUR* | *DE* | *UN(e)* <objet>
> Exemple: Calculer *LE* total *POUR UNE* vente

Ces fonctionnalités sont ensuite regroupées pour former un ensemble plus général. En suivant cette approche, le client peut facilement développer sa compréhension du système.

La troisième étape **planifie les fonctionnalités** qui seront réalisées ensemble. Elle priorise et regroupe les fonctionnalités communes. Ces regroupements doivent être réalisables en moins de deux semaines d'efforts. On détermine réellement les dates buttoirs et les affectations aux développeurs. C'est à la fois une forme de délégation qui permet aux développeurs de planifier les travaux. Pour sa part, le responsable du projet peut se concentrer sur le projet global.

La quatrième étape **conçoit** chaque caractéristique. Cette étape prend en charge la conception d'un groupe de fonctionnalités. C'est à cette étape que les caractéristiques se transforment en classe. L'équipe peut participer à cette activité, afin d'harmoniser la conception.

La cinquième étape **réalise** chaque classe incluse dans une fonctionnalité. Elle se fait subséquemment à la conception. C'est à cette étape que le code est construit, inspecté, testé et intégré. Le propriétaire de la classe se charge de ses travaux.

FDD définit plusieurs rôles. En particulier au niveau des développeurs, il y a les chefs programmeurs et les propriétaires de classe. Les chefs programmeurs se voient confier le développement d'une fonctionnalité, ce qui peut demander l'implication de plusieurs classes. Il rassemble les propriétaires des classes impliquées pour former une équipe fonctionnelle. Ce sont eux qui implanteront les différentes classes pour les besoins de la fonctionnalité.

Mis à part l'objectif de réaliser une fonctionnalité en moins de deux semaines, il n'y a pas de notion de temps fixe (time-boxing), d'itération ou de version. L'équipe de développement est en perpétuel mouvement. Ce sont les groupes de fonctionnalités qui forment l'ampleur et la durée d'une livraison.

La gestion du changement se fait par l'ajout de nouvelles fonctionnalités. Dépendamment de l'ampleur (+10%) de ces ajouts et modifications, la planification du projet peut demander d'être révisée [Highsmith 2002a].

Cette méthode accorde beaucoup d'importance à la modélisation. Elle préfère démarrer plus lentement que d'autre méthode, afin de construire un meilleur modèle général. Elle s'oppose à l'approche de XP sur ce point. Cette méthode a fait ses preuves sur de gros projets. Il est intéressant de noter que TogetherSoft, entreprise de Peter Coad, a été acheté par Borland en 2002. Cet outil de modélisation UML, permet une intégration du design et de la génération de codes, favorisant ainsi l'agilité tout en conservant une documentation système à jour.

3.4.4 Adaptive System Development

Cette méthode a été développée par Jim Highsmith, elle s'inspire de la théorie de la complexité et que tout est un apprentissage. Comme il n'y a pas de pratique formellement décrite, ASD est un gabarit. Cette méthode est une évolution des approches par prototypage (RAD) des années 1980, que son concepteur a largement utilisé [Highsmith 2001].

Avec une approche traditionnelle, il est mal vu de déroger de sa planification. On considère ce comportement comme une incompétence à planifier. Cette culture ne favorise pas l'intégration des nouvelles connaissances, qui changent les perceptions d'origine. Nous sommes aveuglés par la précision de la planification et oublions que les produits évoluent à partir de peu de planification et de beaucoup d'apprentissage. Même s'il est difficile de prévoir les choses, cela n'implique pas qu'il est difficile de progresser [Highsmith 2000].

ASD accepte que les gens ne soient pas infaillible et que le plan ne doit être vu que de manière spéculative. La théorie de la complexité aide à comprendre l'imprévisibilité des projets de développements, qui sont des processus empiriques. ASD cherche l'équilibre entre le chaos et l'ordre, qui a été défini par Dee Hock comme « Chaordic» [Highsmith 2002b].

ASD intègre une culture de changement basé sur trois thèmes : spéculer, collaborer et apprendre. Spéculer laisse le temps d'explorer et permet de dévier du plan sans inquiétude. Cette facette reconnaît la complexité des problèmes et encourage l'exploration et l'expérimentation. La collaboration est nécessaire pour l'évolution de solutions complexes,

qui exigent beaucoup de connaissances diversifiées et qui ont besoin d'être mises en commun.

L'apprentissage résulte de l'expérimentation des connaissances. C'est aussi une forme d'humilité qui nous rappelle que nous ne sommes pas infaillibles. Les utilisateurs fournissent les retours d'expérience pour alimenter le cours du développement.

Il n'y pas de pratiques concrète, mais un cadre général qui comporte quelques valeurs qui constituent les fondements de ASD [BI 2001] :

- Focaliser sur une mission : La mission est le guide du projet. Comme les spécifications ne sont pas écrites, la mission oriente le projet. Elle détermine les limites et non l'objectif.
- Basé sur des composants: Avec un objectif de résultats et non de tâches, les composants sont les éléments livrables du projet. Un composant peut prendre la forme d'une fonctionnalité ou d'un document.
- Itératif : En suivant un modèle itératif incrémental, le produit évolue à partir d'apprentissage. Chaque cycle de livraison ajoute des connaissances qui sont réintroduites dans le projet pour augmenter la satisfaction du client.
- Bloc de temps (Time-boxed) : Les blocs de temps sont souvent utilisés négativement, mettant de la pression et nuisant à la collaboration et à la qualité. Pour ASD, c'est le moyen utilisé pour forcer des décisions difficiles et marquer un temps d'arrêt pour réexaminer la mission du projet.
- Gestion des risques: Étant donné la nature des projets soumis à cette approche, les risques sont analysés dès le départ. Cette gestion peut mettre un terme au projet.
- Tolérance au changement : Cette ouverture est considérée comme un avantage concurrentiel et le résultat d'un apprentissage.

3.4.4.1 Cycle de développement de ASD

Différentes étapes du cycle de développement se retrouvent dans les trois grands thèmes : Spéculer, collaborer et apprendre.

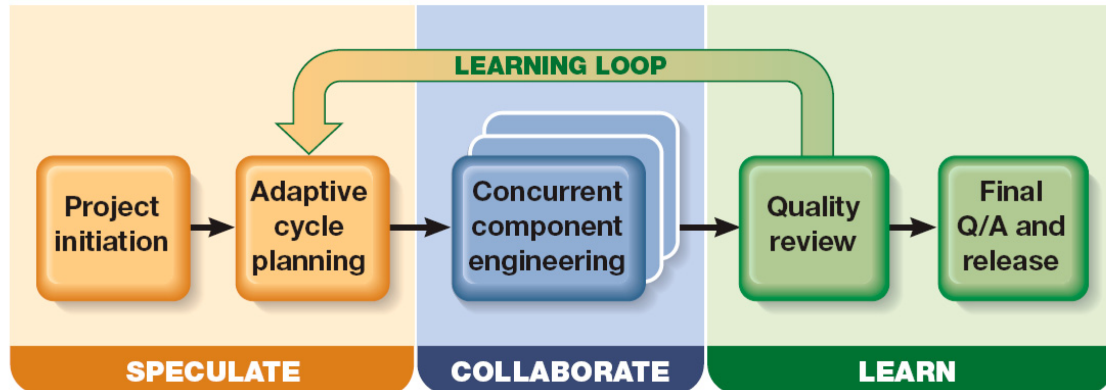


Figure 3-18 : Les grands thèmes et les étapes de ASD

[Tiré de Highsmith 2000]

Initiation – Cette étape détermine l'objectif du projet, les intervenants et les limites du projet. Elle s'appuie sur un atelier de développement commun avec le client (« joint application development » nommé JAD) pour faire ce premier exercice. Cette étape n'est réalisée qu'au départ du projet.

Planification – Cette étape demande de découper le projet en blocs de temps (Time-box), afin de planifier les cycles de livraison. Elle prévoit les ressources nécessaires et les composants inclus dans chaque cycle. Un cycle peut durer de deux à quatre semaines. Chaque cycle possède une mission et doit présenter des fonctionnalités utiles au client.

L'ingénierie simultanée – Cette étape réalise concrètement le produit. Elle nécessite la collaboration des différents intervenants.

Collaboration – Cette étape encourage les intervenants à collaborer. ASD est ancré sur les relations qui sont à la base des organisations. L'équipe de développement est invitée à adopter des pratiques favorisant cette collaboration, tel que la programmation en pairs et le partage du code.

Revue de qualité – Cette étape est le moment pour tirer des leçons et de conserver l'attention sur la mission du cycle. Elle révise la qualité suivant plusieurs aspects. L'appréciation du client se fait comme une approche marketing avec un groupe pilote. La révision technique vérifie le design et le code. Les pratiques de l'équipe sont analysées : ce qui fonctionne bien et ce qui doit être corrigée. Finalement, on y fait le statut du projet pour juger de son avancement et réviser le plan.

Rencontre finale et livraison – Cette étape termine le projet et finalise le livrable. La revue finale de projet aidera à améliorer les prochains développements.

La culture d'optimisation, tel que le CMM, catégorise souvent les choses en noir ou blanc tandis que la culture adaptative reconnaît les zones grises. Cette culture reconnaît que le succès émerge des différents essais, qu'ils soient bons ou pas. Les projets de développement logiciel sont non-linéaires et imprévisibles. Passer à une culture adaptative, outille l'organisation pour le future, plutôt que de l'optimiser sur son passé [Highsmith 2000].

[Page Blanche]

Chapitre 4: Implantation d’une approche agile

Ce chapitre présente la démarche et les résultats de l’implantation d’une approche agile en entreprise. On y décrit la situation et les motivations d’origine. À partir de ces informations, on fixe les objectifs d’améliorations souhaitées. Nous élaborons une stratégie d’implantation, afin de gérer ces changements. Finalement, nous présentons la situation après l’implantation et son adéquation aux approches agiles. Chacune des sections suit les thèmes de l’équipe, du produit, du processus et du client.

L’ensemble de ce chapitre discute d’une situation précise. Ce contexte est spécifique à l’organisation hôte du projet et ne doit pas être considéré comme étant généralisé ou une référence. C’est le témoignage d’une situation réelle, potentiellement transposable à d’autres organisations.

4.1 Contexte d’origine

Le contexte original décrit la situation vécue en 2001, ce qui explique pourquoi nous en parlerons au passé. L’expérience d’implantation a été menée dans une équipe d’entretien et de développement interne, ayant le mandat de supporter les systèmes d’informations d’ordre corporatif.

4.1.1 Description de l’équipe à l’origine

En 2001, l’équipe était composée de cinq personnes : deux programmeurs, un analyste, un administrateur de base de données et un chef d’équipe. Suite aux difficultés de l’organisation, l’équipe avait perdu près de la moitié de ses effectifs au cours des trois dernières années. Le moral de l’équipe était éprouvé. Les gens devaient reprendre les dossiers abandonnés, causant des surcharges sur le personnel restant. Malgré tout, le personnel fournissait un effort constant et considérable.

Le département d’informatique était situé à l’écart. Chaque employé avait son bureau individuel et de bonnes conditions de travail. Certains membres de l’équipe s’impliquaient aussi dans les activités sociales et syndicales.

Depuis plusieurs années, l'équipe se rencontrait moins d'une fois par mois. Les gens interagissaient peu entre eux, car les systèmes étaient cloisonnés. Seuls les projets d'une certaine envergure, telles les mises à jours technologiques, exigeaient de coordonner les efforts.

4.1.2 Description du produit à l'origine

Il y a plusieurs systèmes, de différentes envergures, développés par l'équipe. Le principal CDFI[7], était composé de centaines de modules sur la plate-forme Oracle et servait à toute l'organisation. Sa mission consistait à facturer la clientèle à partir du temps déclaré par les employés. Il était utilisé par tous les départements pour leur suivi des coûts de projet. Lors de sa création, ce système a été développé en suivant une approche traditionnelle et était bien documenté.

Les plus petits systèmes étaient sur des bases de données MS Access, chacun étant utilisé par peu de gens. Moins complexes, ils supportaient des opérations de gestion départementale. Ce type de système était souvent de petite taille. Ces systèmes étaient peu documentés, la plupart étant contenue dans les commentaires du code, ce qui rendait la documentation pratiquement inexistante. Les utilisateurs étaient la principale source de référence.

4.1.3 Description du processus à l'origine

Théoriquement, l'équipe suivait la méthode Productivité PLUS acquise dans les années 1990. Concrètement, nous avions les quatre tomes qui décrivaient la méthode, mais personne n'en avait réellement pris connaissance. Aucune formation ou rappel n'avait été dispensé. Nous produisions le minimum d'éléments pour livrer.

Les demandes se faisaient verbalement ou par écrit. Elles étaient réalisées au gré des opportunités, sans aucun contrôle particulier. Les étapes suivaient un modèle en cascade : analyse – développement – tests – implantation.

[7] CDFI est l'acronyme de Client, Dossier, Facturation, Intervention.

Nous avions un document sur les règles organiques et de bonnes pratiques qui rendaient tout de même l'ensemble efficace et sécuritaire.

4.1.4 Description de la clientèle à l'origine

Les systèmes étant développés pour les besoins de l'organisation, l'ensemble des clients était des employés.

Même au niveau de la direction nouvellement en place, aucun utilisateur n'avait de vision d'ensemble du processus d'affaire couvert par le système CDFI. Chaque utilisateur ne connaissait que la partie qui supportait sa tâche. Cette vision partielle limitait les demandes d'améliorations, le système était considéré inchangeable.

Il n'y avait pas de moyen établi pour faire suivre les demandes. Les pilotes de système étaient informels. Les demandes d'améliorations étaient rares, ce qui limitait nos contacts avec les utilisateurs. De plus, lorsqu'elles survenaient, les efforts de réalisation étaient alourdis, conséquence de la reprise de dossiers des collègues ayant quitté. Il fallait entreprendre une analyse avant de pouvoir estimer l'ampleur du travail demandé. Ce qui aurait pris quelques minutes par le concepteur, prenait quelques jours. L'ouverture aux changements était plutôt limitée.

4.2 Analyse du contexte et objectifs d'amélioration

Suite à la description du contexte d'origine, cette section analyse les différentes améliorations souhaitables. Nous vérifierons quelle approche agile correspond aux améliorations souhaitées. L'objectif de cette démarche est d'implanter une approche agile, pour vérifier si elle peut convenir à une organisation comme la notre.

4.2.1 Analyse du contexte

4.2.1.1 Analyse du contexte de l'équipe

Suite aux différents départs, l'équipe est composée des 5 développeurs les plus expérimentés. Les gens ont réalisé plusieurs systèmes et sont autonomes. Nous avons une petite équipe composée de gens d'expérience et débrouillards.

L'équipe est soumise à peu de contrôle externe. Elle est autonome et peut déterminer quels sont les meilleurs moyens à prendre pour accomplir les tâches. Nous sommes toutefois soumis à quelques contrôles du Vérificateur Général du Québec (VGQ) pour les développements logiciels. L'équipe peut changer ses façons de faire, tout en rencontrant les exigences corporatives.

Les gens travaillent dans un environnement syndiqué, confortable et stable. Ils ont peu de craintes et de réprimandes. L'organisation a une culture de collaboration et non de répression. Ces éléments sécurisants favorisent des échanges ouverts. On peut exposer ses problèmes pour obtenir les conseils de nos coéquipiers.

Tous les membres de l'équipe ont des bases techniques, qui constituent un avantage lors des échanges avec les clients. Ces connaissances permettent de proposer des solutions facilement réalisables techniquement. Les développeurs sont des généralistes qui connaissent les besoins et proposent des solutions livrables rapidement.

Les rencontres d'équipes sont rares et peu stimulantes. Les gens sont dispersés dans différents bureaux et travaillent de manière individuelle. Les échanges se font souvent par écrit. Il serait préférable d'augmenter la fréquence des rencontres et travailler dans un environnement ouvert, minimisant les efforts pour avoir une meilleure communication.

4.2.1.2 Analyse du contexte du produit

Les produits sont réalisés en étroite collaboration avec les utilisateurs et répondent adéquatement à leurs besoins. Les évolutions technologiques et fonctionnelles requièrent régulièrement des changements. L'équipe adapte constamment le système en fonction de leurs demandes. Le système est continuellement adapté aux besoins.

Les systèmes sont développés sur des plates-formes hétérogènes. Cette diversité demande des connaissances dans différentes technologies et la grosseur de l'équipe permet difficilement de diversifier nos expertises. Une rationalisation des technologies serait souhaitable, afin de diminuer les risques technologiques, accélérer nos livraisons et maximiser nos investissements.

L'équipe suit une série de règles fonctionnelles qui constituent les standards techniques. Bien que toujours appliquées, il y a longtemps qu'elles n'ont pas été révisées. Afin de poursuivre leurs utilisations, une actualisation et une formation de mise à jour seraient souhaitables.

La documentation est une part importante du produit. Pour les systèmes les plus importants, elle est très détaillée, mais périmée. Elle a été produite lors de la création des systèmes avec des outils qui ne sont plus disponibles, ce qui la rend lourde à mettre à jour. Pour les plus petits systèmes, cette documentation est souvent incomplète ou inexistante. Il serait souhaitable de motiver les développeurs à mieux documenter leur système, afin de conserver une mémoire corporative. Ces efforts doivent être proportionnels à l'importance du système.

Le code source des systèmes est partagé par toute l'équipe. Une discipline manuelle permet le contrôle des sources. En procédant ainsi, il est possible d'oublier ou d'écraser des codes sources. Il serait bon d'utiliser un outil spécialisé dans le contrôle des codes sources, afin de journaliser les modifications apportées au produit.

Il n'y a pas de gestion de la configuration. Les mises en place sont faites manuellement et aucune trace supplémentaire n'est conservée. Nous pouvons difficilement relier les problèmes engendrés par le déploiement d'un module. À la demande du VGQ, une amélioration de la « traçabilité » des déploiements, est demandée.

4.2.1.3 Analyse du contexte du processus

Le processus des années 1990 est comme un fossile, on en trouve des traces sans en avoir d'exemple concret. Avec le temps et le roulement du personnel, le processus a été graduellement abandonné. Les gens n'ayant pas l'obligation de l'appliquer, on délaissa les

tâches bureaucratiques. Au point où nous en sommes, tout est à construire. Il est souhaitable de remettre en place un processus officiel pour encadrer les développements et l'entretien.

Les nouvelles applications développées suivent une approche en cascade. Les développeurs considèrent que l'on doit prévoir l'ensemble des fonctionnalités au départ. Ils procèdent à des analyses détaillées avant de commencer les travaux. Les livraisons peuvent être faites par phase, mais on cherche à minimiser les changements. Il arrive souvent que le client se démotive et ne soit plus disponible, ce qui allonge les délais et laisse des projets ouverts très longtemps pouvant entraîner d'autres complications. Considérant la modeste grosseur de ces applications, il serait possible d'améliorer les délais de livraison des premières fonctionnalités, pour bien canaliser les efforts et assurer la participation du client.

L'entretien est la principale activité de l'équipe, les demandes sont très ciblées et peu de modules sont touchés à la fois. Dans ce contexte, le processus se résume souvent à une seule cascade. Puisque plusieurs modifications sont apportées au système durant sa vie, la réfection de code et l'utilisation de moyens facilitant les tests seraient souhaitables.

Il n'y a pas de registre officiel des demandes, ce qui complique la planification et la priorisation des améliorations apportées aux systèmes. Il est difficile de regrouper des demandes apparentées. Il serait souhaitable de centraliser les demandes, afin de les gérer.

Il y a peu de suivi au cours des développements. Lors des rencontres, les développeurs évaluent arbitrairement un pourcentage de progression. Ils avancent un pourcentage, mais ne tiennent pas compte de certaines étapes. Il n'y a pas de rapport fait au supérieur ou à un comité, ce qui diminue l'importance de cette pratique. L'utilisation de mesures objectives serait souhaitable pour établir un suivi uniforme et rapporter la progression. De plus, il serait bien d'établir une convention qui détermine l'accomplissement complet (ex : spécification 20%, code 60%…).

Le processus est soumis à peu d'exigence. La certification ISO-9000 ne couvre pas les pratiques de l'équipe. Seul le vérificateur général du Québec (VGQ) doit émettre son appréciation sur le processus de développement et d'entretien. Il a été noté que le processus

n’est pas documenté et laisse peu de traces. Il n’y a pas de preuve d’exécution des tests, des approbations des pilotes et des graduations. Il est nécessaire de documenter le processus d’entretien et de développement des systèmes. Il faut aussi implanter des contrôles établissant la preuve que les essais ont été réalisés et que le pilote accepte d’implanter les modifications.

4.2.1.4 Analyse du contexte de la clientèle

Les clients sont des employés facilement accessibles et qui collaborent bien. Par contre, il n’y a pas de comptabilisation des efforts. Cette illusion de gratuité ouvre la porte à des débordements de fonctionnalités, coûteuses à réaliser. Il faudrait contrôler les efforts investis. Une table de pilotage pourrait répartir les efforts consentis, afin d’établir la priorité des besoins.

Plusieurs employés sont expérimentés et aiment à améliorer la productivité de l’organisation. Ils possèdent des connaissances tacites qu’il est difficile de documenter. En utilisant une approche qui les implique davantage et les sollicite régulièrement, nous augmentons les chances de faire ressortir les éléments importants.

Les responsables de produits ne sont pas clairement établis. En les nommant, ils pourraient déterminer les améliorations du système, former les utilisateurs et approuver les demandes d’accès. Il faudrait déterminer les propriétaires d’applications et définir leurs rôles et responsabilités.

Les utilisateurs connaissent peu l’ensemble des fonctionnalités du système principal CDFI. En connaissant mieux le système, ils pourraient contribuer à l’améliorer. Il serait souhaitable d’offrir des formations continues, ce qui motiverait l’entretien de la documentation.

4.2.2 Objectifs d'amélioration

Les caractéristiques de l'équipe favorisent l'agilité. Nous avons six généralistes expérimentés qui collaborent bien. L'équipe est autonome et soumise à peu de contraintes. Elle peut décider des moyens à prendre pour accomplir sa mission. Nous faisons moins d'une rencontre d'équipe par mois, donc nous pourrions en augmenter la fréquence et établir un ordre du jour récurrent. Le personnel est dispersé, alors nous allons relocaliser les espaces de travail pour faciliter la communication.

Le système est continuellement adapté aux besoins des utilisateurs. La standardisation de la plate-forme pourrait réduire les risques et accélérer les livraisons. Dépendamment de l'ampleur, nous pourrions réduire de quinze minutes à une heure les efforts requis pour les déploiements en automatisant certaines manipulations. Nous définirons mieux les éléments documentaires à produire et leur contenu. La procédure de gestion des codes sources doit être plus sécuritaire et « traçable » pour prévenir les erreurs de manipulation.

Le processus en place est au seuil minimal, se résumant souvent à une seule cascade. Avec le temps, il fût oublié. Nous devrons remettre en place un processus officiel et le documenter. Pour les nouveaux systèmes, nous réduirons le temps de livraison des premières fonctionnalités à moins d'un mois. Nous devrons répertorier toutes les demandes d'amélioration, afin de mieux planifier les travaux. Nous validerons la conformité des livrables avec les attentes des clients.

Les relations avec notre clientèle sont bonnes et les experts sont facilement accessibles. Nous devons déterminer les propriétaires d'applications et mettre en place une table de pilotage pour prioriser les demandes. Les pilotes doivent être mieux informés de leur rôle dans le processus et trouver des moyens pour augmenter leur implication. Finalement, des formations continues devraient être offertes, afin de partager les connaissances et assurer l'utilisation appropriée des systèmes.

4.2.3 Méthodes candidates

En fonction du contexte et des objectifs d'amélioration, il faut déterminer quelle méthode est préférable pour notre implantation. J'ai vérifié plusieurs méthodes, mais je me suis limité à décrire les constatations sur Scrum, Extreme Programming (XP) et Crystal Clear.

Les trois approches agiles se recoupent sur des notions similaires, pouvant s'appliquer dans notre contexte. Elles s'adressent à des équipes de petites tailles, travaillant dans un espace commun. Elles demandent l'implication des clients pour prendre des décisions. La gestion des requis est comparable d'une méthode à l'autre, en prenant en charge un sous-ensemble de tâches à réaliser dans un délai prescrit, révisé cycliquement. La communication est au centre de chacune des méthodes.

Cependant, elles comportent toutes le même problème. Nos demandes d'entretien se font lentement. Il serait difficile de constituer une liste pouvant soutenir un projet tel que présenté dans les méthodes agiles. Toutefois, en considérant que chaque demande constitue un ajout de fonctionnalité, cela peut équivaloir à une nouvelle itération menant à une nouvelle version du produit. Même dans notre situation, les tests automatisés et une architecture évolutive sont de bonnes pratiques à implanter.

4.2.3.1 Adéquation de extreme Programming

Comme son nom l'indique, Extreme Programming (XP) est une méthode extrémiste. Cette méthode précise une douzaine de pratiques à implanter. C'est la méthode qui se préoccupe le plus de l'excellence technique. Elle comporte beaucoup d'impact sur l'organisation du travail.

C'est une méthode difficilement applicable dans un milieu syndiqué, qui suit une structure traditionnelle en ce qui concerne les définitions de tâches. Cependant, il est possible de classer les pratiques en trois groupes : les pratiques individuelles, d'équipes et de gestion. Les pratiques individuelles, comme l'automatisation des tests, sont applicables. Les pratiques d'équipes, tels que le code commun et la programmation en duo, sont partiellement applicables. Les pratiques de gestion, comme le calcul de la vélocité, sont

difficilement applicables, mais peuvent s'adapter. Nous ne pouvons appliquer intégralement cette méthode, mais plusieurs pratiques peuvent en être extraites.

4.2.3.2 Adéquation de Scrum

Scrum s'intéresse principalement à la gestion de projet. Elle propose une excellente approche de la gestion des fonctionnalités pour composer le produit, elle suit des cycles mensuels de rencontres avec ses clients et elle gère les risques avec des suivis quotidiens. Au niveau de la réalisation technique, cette méthode reste ouverte et peut se greffer à XP.

Dans notre cas, les démonstrations mensuelles sont peu envisageables. La nature de nos demandes concerne rarement une ensemble de fonctionnalités cohérentes, tel que présenté par Scrum. Cette disparité multiplie les pilotes qui sont peu intéressés à connaître l'avancement des demandes des autres. Nous devons traiter et présenter séparément chaque demande sans suivre une période mensuelle.

Une autre pratique clef de Scrum, que nous ne pourrions suivre, est les rencontres journalières. L'équipe n'étant pas entièrement dédiée au développement, nous comptons peu de progression quotidienne. Quotidiennement, elles seraient trop fréquentes et s'avèreraient inutiles par contre, nous pouvons retenir de cette méthode, les critères de sélection des fonctionnalités à produire et les questions à poser au cours d'une rencontre de suivi journalier.

4.2.3.3 Adéquation de Crystal Clear

La famille Crystal est peu contraignante et ouverte. Crystal Clear est la méthode pour notre taille d'équipe et nos types de projets. Elle propose des propriétés et des pratiques de base qui seront ensuite adaptées selon leur pertinence. Elle offre l'ouverture pour ajouter nos obligations corporatives et par la suite s'adapter à notre cycle de livraison. C'est avec cette méthode que nous avons tenté notre implantation agile.

La grille suivante présente un rappel des propriétés, stratégies et techniques de Crystal Clear. Chacune d'elles comporte un commentaire sur la possibilité d'implantation. C'est à partir de ce plan que nous avons démarré l'implantation.

Propriétés	**4.2.3.4 Commentaire sur l'implantation**
Livraisons fréquentes	Livrer entre une semaine et un mois.
Réflexion d'amélioration	Aborder le sujet au cours des rencontres.
Communication osmotique	L'aménagement des lieux peut être modifié.
Sécurité personnelle	Les conditions de travail sont stables et sécurisantes.
Concentration dédiée	Au besoin, aménager des espaces de travail « isolés ».
Accès facile aux utilisateurs experts	Utilisateurs internes, toujours disponibles.
Environnement technique favorable	Implanter certaines pratiques, telle la gestion des sources.
Stratégies	
Exploration 360°	Preuve de concepts et discussions en équipe.
Victoire rapide	Viser petit et concret.
Squelette marchant	Notre plate-forme nous aidera.
Ré-architecture incrémentale	Appliquer une approche plus modulaire et dynamique.
Radiateurs d'information	Zone d'affichage des progrès.
Techniques	
Façonnement de la méthode	La documenter, l'analyser et l'améliorer.
Atelier de réflexion	Tenir des rencontres dédiées au sujet.
Blitz de planification	Rencontrer les pilotes pour se coordonner.
Estimation Delphi	Applicable au besoin.
Rencontre journalière	Nos rencontres seront aux deux semaines.
Conception interactive	Impliquer d'avantage les utilisateurs.
Processus miniature	Applicable au besoin.
Programmation côte-à-côte	Applicable au besoin.
Tableau de progression	Prévoir une adaptation.

Tableau 4-9 : Propriétés, stratégies et techniques de Crystal Clear

4.3 Gestion du changement

Plusieurs changements sont souhaitables. Crystal Clear semble la meilleure méthode pour nous. Les approches agiles introduisent des pratiques, mais surtout un changement culturel. Cette section présente la stratégie d'implantation suivie par notre équipe.

4.3.1 Démarche d'implantation

L'objectif fondamental de l'implantation est de mieux travailler en fonction des ressources disponibles. Dans un contexte de rationalisation de ressources, l'optimisation des processus

est un beau projet théorique. Concrètement, nous avons peu de temps à y consacrer. Une intégration graduelle et soutenue est préférable à une implantation drastique et risquée.

Pour se faire, deux stratégies ont été fusionnées. Celle proposée avec Crystal Clear, qui intègre les éléments graduellement en fonction des apprentissages de l'équipe [Cockburn 2004]. Puis, celle présentée dans un article du HBR constituée de huit étapes pour gérer le changement en entreprise [Kotter 1995]. Nous décrivons dans cette section chaque étape et comment elle a été appliquée.

4.3.1.1 Présenter l'urgence d'intervenir

Notre recul méthodologique nous prouvait l'importance d'intervenir. Tout le monde critiquait la situation, le redressement était nécessaire. Les efforts pour intervenir étaient justifiés.

4.3.1.2 Composer une coalition pour mener le changement

Obtenir l'appui des décideurs était simple, car l'organisation du travail est une responsabilité du groupe. Nous sommes autonomes pour choisir nos pratiques. C'est un projet peu coûteux mais, qui peut amener d'importants bénéfices. Dans notre situation, l'exercice ne peut être que bénéfique.

4.3.1.3 Créer une vision pour guider les efforts

Nous avons validé que l'équipe et les types de projets entrent dans les limites de la méthode. Crystal Clear propose des pratiques de base constituant la vision pour une première phase.

4.3.1.4 Communiquer la vision

Cette vision a été communiquée par des formations magistrales et des séances de travail de groupe. J'ai présenté à plusieurs occasions des documents sur les approches agiles, leurs avantages et les opportunités d'appliquer certaines pratiques.

4.3.1.5 Impliquer les gens et les inciter à intervenir

La démarche doit débuter avec des gens conciliants et motivés. Ceci permet de mettre à l'épreuve le changement et éliminer certains obstacles. De cette manière, on sollicite d'autres personnes à devenir des agents de changement. Les pilotes ont été rapidement sensibilisés.

4.3.1.6 Planifier pour créer de petites victoires

Il est préférable de présenter les éléments lorsqu'ils sont pertinents et qu'une situation propice se présente. En visant l'intégration par la pratique, la concrétisation est plus rapide et l'ampleur du projet global est moins imposante. L'application graduelle des outils et des pratiques permettait à l'équipe d'intégrer les changements un à la fois, constituant ainsi une série de petites victoires.

4.3.1.7 Consolider les améliorations et engendrer d'autres changements

Cockburn suggère de conduire de façon cyclique des ateliers de réflexion pour évaluer ce qui peut être amélioré et ce qui sera ajouté. Ce moment de réflexion permet de consolider les changements et planifier les prochaines transformations. Chaque nouvelle pratique sera évaluée en réunion.

4.3.1.8 Institutionnaliser la nouvelle approche

Lorsque la démarche est intégrée, elle peut être documentée et appliquée à d'autres situations. Toutefois, le processus n'est pas figé pour autant. Les ateliers de réflexion sont toujours pertinents pour assurer la viabilité du processus et apporter les révisions.

4.3.2 Chronologie des événements

Cette section résume les événements importants qui ont joué un rôle dans la transition des pratiques.

Au début de l'automne 2002, nous avons fait une rencontre pour schématiser le processus existant et trouver des pistes d'améliorations. Sur papier tout semblait conforme, mais il n'était pas toujours respecté. Le « modèle » était adéquat, mais l'équipe s'entendait pour

dire que nous pouvions faire mieux. Nous manquions de rigueur dans l'application de notre processus.

Suite à ce constat, nous avons développé une application de suivi des demandes (DSI), afin de mieux les gérer. Cette application est une simple liste qui nous permet de les prioriser et les affecter. Les affectations constituent l'agenda des développeurs et les demandeurs peuvent suivre la progression des travaux. Il nous permet de conserver une trace des approbations des pilotes, répondant ainsi à une recommandation du VGQ.

À la fin de l'automne 2002, l'équipe a déménagé. La disposition initiale de l'équipe n'était pas très intéressante et ne permettait pas un environnement favorable à la collaboration. Chaque personne était isolée dans son bureau, à une distance trop importante pour tenter une simple discussion à voix haute. Un déménagement temporaire, permit la réorganisation des bureaux dans un local exclusivement occupé par notre équipe. Chaque développeur conserva un espace individuel. On aménagea une table de rencontre dans un espace libre. Ce fût l'occasion de mettre à l'essai les principes de communication de Cockburn. L'information circulait très bien mais, la table de rencontre était dans le même espace, ce qui était très incommodant lorsque le sujet ne concernait pas tout le monde. Comme cette situation était temporaire, j'ai noté la correction à apporter.

Au cours de l'hiver 2003, toujours dans le même local, nous avons réalisé un projet de refonte de notre système de facturation. Un comité de pilotage nous a remit une liste de plus de 80 caractéristiques à implanter, ce qui représentait des efforts de près de 100 jours-personnes. Nous avons appliqué les cycles itératifs et la priorisation des besoins avec notre pilote. Nous avons fait des mises en place à toutes les semaines. Nous avons rencontré le pilote plusieurs fois par semaine et sommes arrivés à partager un langage commun. Les nouvelles fonctionnalités ont eu beaucoup de succès auprès des utilisateurs, bref cela a été un beau projet. Par contre, il est rare pour nous d'avoir une liste complète des besoins à produire pour une même application. De ce fait, nous ne pouvons appliquer une méthode agile intégralement. Il faut adapter la méthode au rythme de nos demandes.

Ce premier projet a été l'occasion d'implanter un outil d'automatisation de tests unitaires (Ut-pl/sql et Ounit). Implanter un tel outil demande des efforts, mais ils sont récompensés avec le temps. Nous avons implanté les tests unitaires seulement aux endroits stratégiques.

À l'hiver 2004, nous avons réintégré nos locaux permanents. Cette fois, nous avons plus de latitude sur l'aménagement des lieux. Des paravents ont été disposés de manière à maximiser l'entrée de lumière naturelle et laisser l'espace dégagé. Chaque développeur conserve son espace personnel et peut questionner ses collèges sans se lever de son siège. Un bureau fermé est dédié aux réunions et ce local est muni d'un téléphone-conférence et d'un ordinateur. L'équilibre entre l'intimité et la collocation est atteint.

Au cours de cette même période, nous avons apporté plusieurs optimisations techniques. Nous standardisons les différents environnements, afin d'éliminer les adaptations lors des déploiements. Nous structurons les « batchs », afin d'automatiser certaines manipulations. Nous en profitons pour améliorer les messages d'erreurs des systèmes et ajouter des surveillances automatisées. Plusieurs petits systèmes ont été transférés vers la plate-forme Oracle pour consolider nos technologies.

Depuis plusieurs mois, nous avions commencé à céduler nos rencontres de suivi aux deux semaines, donc à l'automne 2004 nous avons maintenu ce rythme des rencontres. Les gens font le tour des demandes en cours et on échange sur les techniques. Chacun peut partager ses impressions et aider les autres. Il arrive que deux développeurs passent une matinée à travailler en duo.

À l'hiver 2005, nous avons mis à jour nos règles organiques, afin d'uniformiser notre développement. Avec les nouveaux outils, nous pouvons être plus souples sur certaines règles, ce qui les simplifie. De plus, nous uniformisons la structure d'édition des commentaires, afin de permettre à un outil d'extraction (PL Doc) de produire une documentation similaire à la Javadoc. Nous implantons aussi un Wiki (moin-moin) pour centraliser les connaissances de l'équipe et documenter notre fonctionnement.

Notre procédure de gestion des sources pouvait être erronée. Pour corriger la situation, nous avons implanté un logiciel de contrôle de source (Subversion et Tortoise). Cet outil nous

permet d'identifier tous les éléments réalisés suite à une demande. Cette trace nous permet de répondre aux exigences du VGQ.

4.4 Analyse des résultats obtenus

Nous pouvons désormais établir une liste des changements issus de l'implantation. L'agilité demande un changement de culture qui s'intègre lentement. Il faut revenir sur le message, saisir les opportunités qui se présentent et démontrer les bénéfices. L'effort d'implantation est un travail constant. Nous présentons la situation actuelle en fonction de ce qui a changé.

4.4.1 Changements apportés à l'équipe

L'équipe possédait au départ plusieurs attributs favorisant l'intégration d'une approche agile, soit une petite équipe de gens expérimentés et autonomes. Ces éléments favorables n'ont pas subi de changement.

La disposition des bureaux est ce qui se remarque le plus, nous sommes dans une aire semi-ouverte. Ce modèle facilite la communication et permet à chacun de conserver son intimité. C'est un compromis acceptable, car les gens accordent encore de l'importance à avoir un bureau individuel. La communication entre les individus est importante [Cockburn 2001]. L'information circule plus facilement. L'utilisation de simples outils de communication, tel qu'un tableau blanc, améliore les échanges et nous rend plus efficace.

Nous avons augmenté la fréquence des rencontres, passant ainsi à une rencontre à toutes les deux semaines. Nos espaces de travail sont tous co-localisés dans une aire ouverte, ce qui favorise les échanges quotidiens. Il est ainsi plus facile de rester à l'affût des événements. Nous continuons d'améliorer nos suivis avec notre outil de gestion des demandes.

Nous améliorons nos façons de faire et déterminons quelles solutions seraient intéressantes. Nous partageons un objectif commun d'améliorer notre fonctionnement. L'équipe sait qu'elle est la principale responsable de son succès.

Par contre, il est aussi difficile d'affecter les tâches verticalement, telle que suggéré par XP. C'est-à-dire, prendre en charge une fonctionnalité de la prise des besoins jusqu'à la

livraison. Dans un environnement syndiqué, la convention définit les postes de manière horizontale. On parle ici des opérateurs, des techniciens et des analystes, auxquels s'ajoutent des spécialités qui cloisonnent davantage. Ne pouvant contourner cette contrainte, la séparation des rôles existe toujours. Les échanges directs contribuent à réduire les efforts des transferts de connaissance mais, une même personne ne peut prendre en charge la réalisation complète.

4.4.2 Changements apportés au produit

Au niveau produit, les systèmes ont été peu touchés, car ils étaient déjà de bonnes qualités. L'agilité ne fait pas nécessairement de meilleurs produits, mais les conçoit différemment. Les nouvelles pratiques ont tout de même augmenté la fiabilité et simplifié la gestion des sources, mais ces bénéfices sont liés à l'amélioration de nos pratiques. Les standards de réalisation ont été révisés.

Nous avons amélioré la documentation des systèmes, surtout au niveau de l'architecture mais, il reste encore beaucoup à faire. Désormais, nous ajustons le niveau de documentation requis en fonction de l'importance du système. Nous pouvons identifier les éléments nécessaires en fonction du niveau critique du système. Comme les nouveaux systèmes sont assez petits, cette documentation peut être légère. Cette pratique nous permet aussi de les répertorier dans notre portefeuille applicatif.

L'équipe est invitée à mener des réflexions sur la pertinence des éléments documentaires. En cours de développement, les agilistes préfèrent les conversations directes. En entretien il faut limiter cette documentation et les efforts requis pour l'entretenir, autrement elle risque de devenir désuète. Les outils modernes aident les équipes d'entretien en rapportant les éléments existant d'un système, ce qui allège le travail.

Nous continuons à utiliser l'outil de modélisation d'Oracle pour nos différents diagrammes. Tous les éléments graphiques sont conçu avec cet outil. Le modèle relationnel reste le plus important élément de la documentation. Les dossiers fonctionnels qui renferment les règles d'affaires, sont aussi des éléments documentaires important. Ils conservent l'expertise du

système. Nous avons toutefois révisé leur contenu, en diminuant certaines sections lourdent à mettre à jour pour nos besoins d'entretien.

Avec l'utilisation du système de gestion des demandes, les interventions sur les produits sont « rétractables » et la journalisation est complète. Il est aussi possible d'établir un lien entre le système de demandes et le lancement des applications, qui peuvent informer l'utilisateur des améliorations apportées depuis sa dernière l'utilisation. Cette technique permet d'informer continuellement les utilisateurs au moment opportun et motive l'équipe de développement à mieux rédiger les demandes d'améliorations.

La plupart des méthodes agiles s'adressent aux environnements orientés objets. Dans notre cas, nos systèmes d'informations sont développés avec les produits et la base de données Oracle. Nous n'avons toutefois pas été contraints par cette différence. L'évolution des dernières versions de cette plate-forme se prête bien aux pratiques agiles. De plus, l'administrateur de base de données s'implique dans les développements, ce qui facilite l'intégration des changements.

Nous avons consolidé la plate-forme technologique des petits systèmes, augmentant ainsi la réutilisation et accélérant nos développements. Cette orientation améliore notre expertise et diminue les risques, nous avons moins de langage et d'outils à connaître et à entretenir. Ceci nous a permis de simplifier nos techniques de déploiement en uniformisant la structure des serveurs de nos différents environnements.

4.4.3 Changements apportés au processus

Au départ, je croyais qu'il suffirait de réorganiser la manière d'ordonnancer les tâches et réviser les livrables pour entrer dans le monde agile. Il ne s'agit que d'une parcelle des éléments, l'interaction entre les individus et les communications ont plus d'impact.

Il nous fallait un moyen de gérer les besoins et dans notre cas, gérer les demandes d'amélioration formulées par les pilotes. Nous avons produit une petite application qui permet cette saisie. Ce système nous permet de prioriser les demandes et de planifier les tâches à réaliser. À la manière agile, nous sélectionnons les demandes à produire prochainement. Il est possible de suivre les améliorations apportées aux systèmes. Ce

système conserve les traces et permet de rendre des comptes plus facilement. Du coup, nous nous sommes conformés à une exigence qui visait à journaliser les modifications et les approbations.

Le modèle suivi lors de la réalisation d'une demande, est une adaptation des cycles itératifs. Nous déployons rapidement une version simpliste, sur laquelle nous demandons les impressions du demandeur. Dans notre contexte, nous ne fixons pas de période de temps à nos cycles de réalisation, nous avons plutôt adapté par des rencontres chaînées. À chacune des rencontres, nous convenons de la date de la prochaine rencontre et ce, dans un délai raisonnable. La demande doit avoir suffisamment évoluée pour alimenter le contenu de cette rencontre. Ce cycle itératif, de longueur variable, assure au demandeur qu'il pourra suivre la progression de sa demande.

La réalisation technique a été améliorée par certaines pratiques. Tous les codes sources sont centralisés dans un outil spécialisé et les modifications apportées font référence à leur demande d'origine. Nous avons aussi ajouté des éléments liés au système, tels que les scripts de création et les tests, qui font désormais partie des éléments du système. Ces éléments, auparavant volatiles, sont maintenant considérés comme faisant partie du système et sont conservés.

Les développeurs ont peu d'intérêt pour les méthodologies, cet aspect étant considéré bureaucratique et superflu. En présentant le processus comme une convention [Cockburn 2004], il est plus facile pour l'équipe de se l'approprier. Ce n'est plus la méthodologie d'une grande firme, c'est la méthode adaptée qui convient à l'équipe, à l'organisation et au produit. La description du processus est produite et permet d'être constamment révisée dans le Wiki du groupe. Si nous précisons un élément, tous pourront en prendre connaissance ou se rafraîchir la mémoire. Cette nouvelle approche de documentation nous a aussi motivé à mettre à jour nos règles organiques et à les publier.

L'automatisation des tests n'a pas eu l'accueil attendu. Le système est mature et fiable mais, il est difficile de convaincre de la pertinence de cette pratique. Nous l'avons implanté lorsque la complexité et la possibilité d'erreur étaient forte, ce qui nous laisse peu de cas.

4.4.4 Changements apportés avec nos clients

Nous avons amélioré nos échanges avec nos clients. En livrant souvent des petites améliorations, nous avons développé une complicité dans le but d'atteindre un objectif commun. Cette proximité, entre leur formulation d'une demande et l'atteinte de résultat, augmente le sentiment de propriété du système. Les clients sont plus ouverts aux discussions et il est plus facile de négocier des solutions plutôt que considérer leurs demandes irréalisables. Les clients connaissent mieux la vision des développeurs et sont heureux de partager un langage commun. Les utilisateurs réalisent plus facilement que le système est adapté à leurs besoins.

On entretient plus facilement la motivation, car les rencontres sont régulières. Les gens sont mieux informés de nos façons de faire et en sont respectueuses. En leur donnant le sentiment d'être pris en charge dans un processus établi, ils sont en confiance et plus disponibles.

Lors des rencontres de prise des besoins, nous utilisons des moyens simples qui permettent aux utilisateurs d'interagir. Chaque développeur a un tableau blanc dans son bureau, ce qui facilite les discussions techniques. La salle de rencontre est équipée d'un ordinateur et d'un tableau blanc. Le système de visioconférence est aussi très efficace pour discuter avec les utilisateurs distants.

Chaque système est sous la responsabilité d'un pilote. Ce responsable canalise les demandes, effectue les tests fonctionnels et approuve les mises en production. Il s'occupe de la formation des utilisateurs. Il nous reste à tenir des rencontres de pilotage, afin que chaque responsable expose ses besoins. Cette pratique de contrôle pourra prioriser les demandes et permettra d'affecter plus efficacement nos développements.

4.5 Atteinte de l'agilité

En général, l'implantation d'une approche agile a été satisfaisante et elle continue de s'améliorer. Certaines pratiques n'ont pas adhéré par manque de pertinence, de maturité des outils ou parce que les gens n'étaient pas encore prêts.

Il est difficile de quantifier le niveau d'agilité, car il n'y a pas d'échelle de mesure. Nous pouvons considérer que nous ne sommes pas complètement agiles pour des raisons similaires à celles qui font que les développements « open source » n'en sont pas [Cockburn 2004]. Les méthodes agiles s'adressent principalement à des firmes de consultants, travaillant sur des projets à ressources limitées. Ils doivent équilibrer les fonctionnalités, le temps et le budget. Le contexte de nos projets n'étant pas concerné par ces contraintes, nous ne pouvons utiliser cette appellation.

Toutefois, nous suivons les mêmes valeurs et partageons les mêmes pratiques. Nous valorisons la communication directe et l'ajout concret de valeur au produit. Nous suivons l'approche adaptée suffisamment rigoureuse, pour être dans une zone confortable et sécuritaire. Les utilisateurs savent qu'ils ont une influence sur le système et collaborent très bien avec l'équipe. Pour ces raisons, nous nous qualifions comme étant semi-agile. Notre situation actuelle est meilleure qu'il y a cinq ans et va continuer de s'améliorer.

4.6 Conclusion

Au départ, nous avons constaté que l'équipe a délaissé plusieurs pratiques au fil des ans. Les réductions du personnel ont provoqué une perte d'expertise et un recul méthodologique, ce qui entraîna un impact à long terme.

On doit analyser objectivement son contexte organisationnel, afin de vérifier quels sont les avantages et inconvénients rattachés aux approches agiles. Si elle se juge candidate, l'organisation doit s'assurer qu'elle est disposée à faire les efforts requis pour adopter ses pratiques et sa culture. L'agilité peut être intéressante pour plusieurs raisons, mais n'est pas adéquate à toutes les organisations.

Lorsque la méthode et les pratiques à implanter ont été choisies, il faut déterminer la stratégie d'implantation. Pour une équipe travaillant ensemble depuis longtemps, il est préférable d'apporter les changements graduellement, afin d'éviter une rupture pouvant compromettre le projet d'implantation. En suivant la philosophie agile, il est possible d'implanter graduellement différentes pratiques en commençant par les plus importantes.

Pour notre équipe, le développement de petits systèmes sont les meilleures opportunités pour l'utilisation d'approches agiles. Comme le système doit aider les gens dans leur travail et que ce dernier peut changer facilement, il est préférable d'adopter une approche agile pour faciliter les adaptations.

Les approches agiles apportent une vision différente, qui nous permet d'améliorer nos pratiques et s'adapter à notre contexte. Ce projet nous a permis d'actualiser plusieurs bonnes pratiques négligées au cours des dernières années. Nous sommes les artisans de notre succès, l'équipe doit déterminer la meilleure approche pour elle, afin de mener ses projets à terme et satisfaire sa clientèle.

[Page Blanche]

Chapitre 5: Conclusion

Pour conclure l'ensemble de ce mémoire, je présente une synthèse des approches traditionnelles et agiles, ainsi que les constations faites au cours de ces années de veille sur le sujet. Je présente finalement la conclusion sur ces efforts et ma vision de l'agilité.

5.1 L'offre traditionnelle

Au début de l'aire informatique, les logiciels étaient développés par leur utilisateur bénéficiaire. Avec l'accroissement de leur complexité, la séparation des rôles de développeur et d'utilisateur est apparu. À partir de ce moment, la définition d'un processus a été nécessaire, afin de structurer la démarche. Celle-ci établit qu'il faut comprendre et analyser les besoins pour planifier la réalisation et minimiser les changements.

Cette approche est similaire aux disciplines d'ingénierie traditionnelles, travaillant avec des produits tangibles. Elle sépare la conception des spécifications de la réalisation du produit. Les critères de succès sont généralement évalués en fonction du respect des échéanciers et des budgets.

Le produit, caractérisé par ses fonctionnalités, est une variable moins suivie. Il est possible qu'au terme du projet, le logiciel soit incomplet ou de mauvaise qualité. Si l'on désire améliorer la qualité ou compléter des fonctionnalités, cela implique des dépassements de coûts et d'échéancier. Ces dépassements sont souvent nécessaires, afin d'obtenir un produit fonctionnel.

Malgré les critiques, les approches traditionnelles sont encore les plus populaires. Ce sont de bonnes approches pour les projets de grande envergure et rigoureux. Elles sont complètes et requièrent d'importants efforts de gestion. Pourtant, ces approches ne sont pas adéquates à tous les projets. Les gens aiment ce qui est simple et efficace, ce qui est paradoxal avec ce type d'approche.

5.2 L'offre agile

Le manifeste reste le meilleur résumé pour synthétiser le changement de paradigme des approches agiles. Les approches agiles priorisent les résultats, les communications, la

collaboration et intègrent les changements en cours de projet. Elles exploitent l'aspect intangible des logiciels, spécifique à cette industrie. Avec ces approches, les critères de succès sont différents.

Diverses motivations peuvent faire choisir ce genre d'approche. La rapidité de mise en marché permet de saisir des opportunités d'affaire. Une préoccupation de fournir un produit de qualité adapté au besoin de la clientèle. Même dans un cadre plus traditionnel, elles permettent un meilleur contrôle budgétaire et des échéanciers.

L'aspect intangible du logiciel lui permet une plus grande flexibilité, ce qui diffère des produits traditionnels. Il est possible d'améliorer le produit en cours d'exploitation, ce qui permet de livrer des versions réduites pouvant se compléter par la suite, ce qui accélère le retour sur l'investissement.

Ce changement questionne les critères de succès des projets. Les chiffres présentés touchent le dépassement de coûts et d'échéancier mais, on ne précise jamais si le produit final comporte toutes les fonctionnalités prévues et leur niveau de qualité.

L'agilité priorise la qualité du produit. La collaboration avec le client et le feedback rapide augmente l'adéquation des fonctionnalités aux tâches qu'il supporte. L'objectif de livrer un produit valable au client, est le plus important critère de succès. Les coûts et l'échéancier prévus peuvent dépasser les prévisions mais, dans un tel cas, demeure la volonté du client.

L'agilité permet aussi de prioriser les métriques traditionnelles de coût et d'échéancier. Le découpage itératif permet d'arrêter le projet si désiré, tout en ayant un produit fonctionnel. Le suivi mensuel conseillé, contrôle les coûts et l'échéancier. Un éventuel débordement est ainsi limité à un cycle. En priorisant ces variables, le projet sera un succès mais, le produit peut être incomplet par rapport au plan initial. On aura sacrifié des fonctionnalités pour respecter le budget et l'échéancier.

5.3 Constatations sur l'agilité

L'agilité doit être présentée adéquatement, ces approches sont exigeantes. Elles demandent de la discipline et le courage de se prendre en main. Ce n'est pas toujours ce qui convient

aux gens. Certaines équipes préfèrent suivre un processus établi, sans en garantir le succès. C'est pourtant une belle occasion de permette aux gens de contribuer à l'amélioration des conventions de travail. En s'appropriant le processus, l'équipe peut diminuer ses irritants et augmenter son efficacité.

Du point de vue technique, les outils de développement et les langages ont beaucoup évolué, intégrant le design, le code, les tests et le déploiement. Nous oublions facilement les efforts jadis requis, pour arriver au même niveau de productivité. Les délais entre la conception et la livraison ont beaucoup réduis. La nouvelle génération de développeurs est habituée aux résultats rapides. Cette productivité peut s'avérer frustrante lorsqu'elle est gérée par un processus traditionnel. L'agilité offre une opportunité intéressante pour gérer les projets avec les nouvelles technologies de développement.

Pour les utilisateurs, livrer rapidement est un avantage encore méconnu. Les clients impliqués dans le développement comprennent que les premières livraisons comportent peu de fonctionnalités. Il peut y avoir de l'incompréhension et des déceptions de la part des autres utilisateurs (non impliqués). Ils peuvent considérer que le système n'est pas complet ou assez soigné. Ils ont la sensation que le système est complet et inchangeable. Il faut les éduquer et faire comprendre ce type d'approche. Surtout, informer que si quelque chose ne convient pas, il n'est pas trop tard pour le corriger.

5.4 Questionnement initial

Dans l'introduction de ce mémoire, quelques questions ont été soulevées. Elles ont toutes été répondues à travers les chapitres mais, voici plus précisément les réponses.

Le peu de documentation générée par ces méthodes est-elle suffisante pour l'apprentissage du système par les nouveaux utilisateurs? Cette inquiétude provient de XP qui laisse croire qu'elle n'existe pas. L'ensemble des approches agiles souligne que la documentation doit être produite. Elle peut prendre diverses formes et demeure un élément complémentaire. Avec une approche traditionnelle, on constate souvent que l'étape des tests est écourtée. Avec une approche agile, on peut établir une similitude avec la documentation. Pourtant, il est possible de bien tester avec une approche traditionnelle, comme il est possible de bien documenter pour une approche agile. En révisant l'utilité des éléments produits, il est possible de la simplifier. Une image vaut mille mots. Les représentations graphiques sont aussi de la documentation. Il est possible d'impliquer les utilisateurs dans cette portion du travail.

Les équipes de développement peuvent changer et l'entretien peut aussi être assuré par des consultants externes. Auront-ils assez d'information pour pouvoir répondre rapidement et efficacement aux demandes du client? Cette question rejoint la première, car elle couvre la qualité de la documentation. Par contre, les approches agiles développent en fonction de faire évoluer les systèmes. Les équipes qui se préoccupent de l'excellence technique, devraient laisser un code minimal, propre et bien commenté. Conjointement avec une bonne gestion de la configuration, les tests automatisés facilitent les interventions en alertant rapidement l'état du système.

Peut-on satisfaire aux exigences de la norme ISO-9001 : 2000 en utilisant une telle méthode? Cette norme établit un corollaire entre satisfaire la clientèle et la qualité du processus. Il est possible d'intégrer les exigences de la norme dans un processus agile. Une série de points demandent à être couverts mais, ce n'est pas conflictuel avec une approche agile. La définition du processus décrit ce que l'organisation fait. Ces pratiques peuvent évoluer et être rapportées dans la définition du processus.

Les approches traditionnelles ont encore leur place et demeurent un bon choix pour certaines organisations. Cependant, elles offrent peu de solutions aux petites équipes et restent accrochées à un processus complexe. Elles sont prédictives et l'adéquation des réalisations avec la planification est un gage de succès.

L'agilité propose d'alléger le processus et comporte différents intérêts. Pour l'équipe de développement, c'est une meilleure collaboration avec ses collègues. Pour le produit, c'est un rehaussement de qualité et d'évolutivité. Pour la gestion de projet, c'est le contrôle de l'échéancier et du budget. Pour le client, c'est d'obtenir un système opérationnel valable rapidement. Ces méthodes sont adaptatives et intègrent les changements pour maximiser l'efficacité du produit dans sa mission.

L'agilité élargit l'offre des approches de développement et mérite d'être connue. Elle s'adresse à une catégorie de projet et d'équipe. À mon avis, comme plusieurs mouvements de pensée émergent, elle suivra son cours et les gens y adhéreront au fil du temps. Les approches agiles se comparent à la sensibilisation écologique. Au début, les gens se souciaient peu des problèmes environnementaux promus par une poignée d'écologistes « marginaux ». De nos jours, ils sont devenus un enjeu important.

Je conclus en mentionnant que je suis satisfait du travail accompli et que j'ai la sensation d'avoir fait progresser les choses. J'ai appliqué les éléments découverts au cours de mes lectures et ce, au profit de mon équipe de travail. Je n'aurais pas été en mesure d'implanter ces éléments avec autant de convictions, sans connaître les solides théories supportant l'agilité. Avec un peu de recul, je ne cherche plus à convaincre mais, seulement à présenter l'agilité. Fidèle aux valeurs agiles, je laisse les gens décider de leur orientation. Si cette prise de contact mène à une réflexion qui améliore les travaux d'une équipe, je serai heureux d'y avoir contribué.

[Page Blanche]

Bibliographie

[AA 2005a] Alliance agile. An Agile Roadmap. Sur le site officiel de l'*alliance agile* (http://www.agilealliance.com/resources/roadmap/) consulté en décembre 2005.

[AA 2005b] Alliance agile. Manifesto for Agile Software Development. Sur le site officiel de *l'alliance agile* (http://www.agilemanifesto.org/) consulté en décembre 2005.

[AA 2005c] Alliance agile. Lean Software Development. Sur le site officiel de *l'alliance agile* http://www.agilealliance.com/resources/roadmap/lean/lean_index) consulté en décembre 2005.

[Aberdeen 1995] Aberdeen Group. Upgrading To ISV Methodology For Enterprise. *Application Development Product Viewpoint* Volume 8, Number 17, 1995.

[ADM 1996a] Advanced Development Methods, Inc (ADM). Controlled Chaos : Living on the Edge. 1996. Sur le site officiel de *SCRUM* (http://www.controlchaos.com/download/Living%20on%20the%20Edge.pdf) consulté en novembre 2005.

[ADM 1996b] Advanced Development Methods, Inc (ADM). Controlled-Chaos Software Development. Sur le site officiel de *SCRUM* (http://www.controlchaos.com/download/Controlled-Chaos%20Software%20Development.pdf) consulté en novembre 2005.

[ADM 2003] Advanced Development Methods, Inc (ADM). SCRUM, It Depends on Common Sense. 2003. Sur le site officiel du *Réseau agile canadien* (http://www.agilenetwork.ca/camug-static/pres/CAMUG_021903.pdf) consulté en octobre 2005.

[ADM 2005] Advanced Development Methods, Inc (ADM). Agile Processes - Emergence of Essential Systems. Sur le site officiel de *SCRUM* (http://www.controlchaos.com/download/Emergence%20of%20Essential%20Systems.pdf) consulté en novembre 2005.

[ADM 2005-2] Advanced Development Methods, Inc (ADM). Philosophies of Software Development. Sur le site officiel de *SCRUM* (http://www.controlchaos.com/old-site/philsd.htm) consulté en novembre 2005.

[Alter 2001] Alter, S. Which life cycle, work system, information system, or software? *Communications of the association for information systems*, Volume 7, Article 17, October 2001.

[AM 2006] Agile Montréal. Principes derrière le Manifeste Agile. Site officiel du *groupe des utilisateurs agile de Montréal* (http://www.pyxis-tech.com/agilemontreal/fr/agileprinciples_fr.php) consulté en septembre 2006.

[Ambler 2004] Ambler, S.W. Are you agile or are you fragile? *Conférence du Computer History Museum* (http://video.google.com/videoplay?docid=4909173801395521O2&q=ambler) consulté en avril 2004.

[Ambler 2005a] Site de Scott W. Ambler (http://www.ambysoft.com/) consulté en avril 2006.

[Ambler 2005b] Ambler, S. W. Enterprise unified process (EUP), 2005. Sur le site de *Scott W Ambler* (http://www.enterpriseunifiedprocess.com/) consulté en avril 2006.

[Ambler 2005c] Ambler, S. W. The Agile Unified Process (AUP), 2005. Sur le site de *Scott W Ambler* (http://www.ambysoft.com/unifiedprocess/agileUP.html) consulté en avril 2006.

[Anderson 2005] Anderson, D. J. Stretching Agile to fit CMMI Level 3. *Proceedings of Agile Conference,* Denver July 2005.

[Arsenault 1997] Arsenault, L. *Dictionnaire des compétences TRIMA. Arsenault Formation Carrière, 1997.*

[Augustine 2003] Augustine, S. and Woodcock, S. Agile Project Management, 2003. Sur le site de *CC Pace inc.* (www.ccpace.com) consulté en septembre 2006.

[Beauregard 2006] Beauregard, F. Visual Studio Talk Show : Entrevue sur les approches agiles, 2006. *Site officiel du groupe des utilisateurs agile de Montréal* (http://www.pyxis-tech.com/agilemontreal/) consulter en septembre 2006.

[Beck 1999] Beck, K. Embracing Change with Extreme Programming. *IEEE Computer magazine,* Octobre 1999.

[Beck 2000] Beck, K. *Extreme Programming Explain : Embrace change.* Addison-Wesley, 2000.

[Beck 2005] Beck, K. *Kent Beck (biographie).* Sur le site de *Cunningham & Cunningham, Inc*. (http://c2.com/ppr/about/author/kent.html) consultée en septembre 2005.

[BI 2001] Business Interactif. Extrême Programming - Méthodes Agiles: L'état des lieux, 2001. Sur le site de *Business Interactif* (http://www.businessinteractif.fr/) consulté en février 2004.

[BI 2002] Business Interactif. Méthodes agiles: une réponse à un malaise? 2002. Sur le site de *Business Interactif* (http://www.businessinteractif.fr/) consulté en février 2004.

[Boehm 1988] Boehm, B. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, May 1988, p. 61-72.

[Boehm 1998] Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R. Using the Win-Win Spiral Model: A Case Study. *IEEE Computer,* July 1998, p. 33-44 .

[Boehm 2002] Boehm, B. Get Ready for Agile Methods, with Care. *IEEE Computer*, January 2002, p.64-69.

[Boehm 2004] Boehm, B., Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, Boston, MA. 2004.

[Booch 1995] Booch, G. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1995.

[Brook 1975] Brooks, F.P. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1975 Réédition (8 août 1995).

[Clever 2004] Clever Link. CMMI ou comment maîtriser vos développements, 2004. Sur le site de *Clever Link* (http://www.clever-age.com/veille/clever-link/cmmi-comment-maitriser-vos-developpements-188.html) consulté en avril 2006.

[Cockburn 1997] Cockburn, A. *Surviving Object-Oriented Projects.* Addison-Wesley Professional, 1997.

[Cockburn 2001] Cockburn, A. *Agile Software Development.* Addison-Wesley Professional, 2001.

[Cockburn 2000] Cockburn, A. Characterizing people as non-linear, first-order components in software development. *Proceeding at the 4th International Multi-Conference on Systems, Cybernetics and Informatics*,Orlando, Florida, June, 2000.

[Cockburn 2004] Cockburn, A. *Crystal Clear: A Human-Powered Methodology for Small Teams.* Addison-Wesley, 2004.

[Cohn 2004] Cohn, M.W. Selecting an Agile Process: Choosing Among the Leading Alternatives. *Proceeding of SD Best pratices; conference & expo 2004,* September 21, 2004.

[Cohn 2005a] Cohn, M.W. The Scrum Development Process. Sur le site de *Mountain Goat Software* (http://www.mountaingoatsoftware.com/scrum/index.php) consulté en novembre 2005.

[Cohn 2005b] Cohn, M.W. Project Economics: Selecting and Prioritizing High Value Projects, 2005. Sur le site de *Mountain Goat Software* (http://www.mountaingoatsoftware.com/system/presentation/file/43/SDBP2006_ProjectEconomics.pdf) consulté en novembre 2005.

[Coplien 1994] Coplien, J.O. Borland Software Craftsmanship: A New Look at Process, Quality and Productivity. *Proceedings of the 5th Annual Borland International Conference*, Orlando, Florida, June 5, 1994.

[CTG 1998] Center for Technology in Government. A Survey of System Development Process Models, 1998. Sur le site de *University at Albany* (http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf) consulté en novembre 2005.

[Deming 1982] Deming, W. E. *Out of the Crisis.* MIT Press, February 2, 1982.

[DGI 1990a] Direction générale de l'informatique et le Groupe DMR inc. *Cadre normatif; Guide de gestion de projet.* Recherche et développement Groupe DMR inc. 1987.

[DGI 1990b]Direction générale de l'informatique et le Groupe DMR inc. *Guide de développement de système.* Recherche et développement Groupe DMR inc. 1987.

[DSDM 2005]Site du *consortium de Dynamic Systems Development Method (DSDM) (*www.dsdm.org*).*

[Fowler 2001] Fowler, M. *and Highsmith, J.* The agile manifesto. 2001. Sur le site de *Dr. Dobb's portal (*http://www.ddj.com/dept/architect/184414755 *) consulté en mai 2006.*

[Fowler 2002] Fowler, M. The Agile Manifesto: where it came from and where it may go. Sur le site de *Martin Fowler* (http://martinfowler.com/articles/agileStory.html) consulté en mai 2006.

[Fowler 2005] Fowler, M. The new methodology. Sur le site de *Martin Fowler* (http://www.martinfowler.com/articles/newMethodology.html) consulté en avril 2006.

[Ghezzi 1991] Ghezzi, C., Jazayeri, M. and Mandrioli, D. *Fundamental of Software Engineering.* Prentice Hall, New Jersey, 1991.

[Gilb 1988] Gilb, T. *Principles of Software Engineering Management.* Addison-Wesley, Wokingham, UK, 1988.

[Glassborow 2002] Glassborow, F. Reviews: Questioning Extreme Programming. 2002.Sur le site de *ACCU* (http://www.accu.org/bookreviews/public/reviews/q/q003273.htm) consultée en septembre 2005.

[Graham 1994] Graham, I. *Migrating to Object Technology.* Addison-Wesley, 1994.

[Hartmann 2006] Hartmann, D. and Dymond, R. Appropriate Agile Measurement:Using Metrics and Diagnostics to Deliver Business Value. Sur le site de *Berteig Consulting* (www.berteigconsulting.com/AppropriateAgileMeasurement.pdf) consulté en août 2006.

[Kotter 1995] Kotter, J.P. Leading change: Why transformation efforts fail. *Havard business review,* march-april 1995 p. 59-68.

[Hesse 2003] Hesse, W. Dinosaur Meets Archaeopteryx? or: Is there an Alternative for Rational's Unified Process? *Software and Systems Modeling (SoSyM)* 2003, Vol. 2. No. 4, p. 240-247.

[Highsmith 2000] Highsmith, J. Retiring Lifecycle Dinosaurs. *Software Testing & Quality Engineering,* July/August 2000.

[Highsmith 2001] Highsmith, J. Agile methodologies: Problems, Principles, and Practices. 2001. Site de Cutter Consortium (http://rockfish.cs.unc.edu/COMP290-agile/AgileMethodologiesXP2001.pdf) consulté en janvier 2004.

[Highsmith 2002a] Highsmith, J. *Agile Software Developement Ecosystems*. Addison-Wesley, Boston, 2002.

[Highsmith 2002b] Highsmith, J. What Is Agile Software Development? *CROSSTALK The Journal of Defense Software Engineering,* October 2002.

[IBM 2005] IBM. RUP in the dialogue with Scrum. 2005. Site officiel de *SCRUM (*http://www.controlchaos.com/module/RationalEdge0205.pdf*)* consulté en novembre 2005.

[IBM 2006] IBM. Rational Unified Process. 2006. Site officiel de *RUP* (http://www-306.ibm.com/software/awdtools/rup/) consulté en mai 2006.

[IEEE 2000] Rising, L. and Janoff, N. S. The Scrum Software Development Process for Small Teams. *IEEE software* Juillet/Août 2000.

[Jefferies 1999] Jefferies, R.E. Extreme testing. *Software Testing & Quality Engineering*, March/April 1999, p. 23-26.

[Kilpatrick 2003] Kilpatrick, J. Lean Principles. *Manufacturing Extension Partnership,* Utah, 2003.

[Kumar 2005] Kumar, D.R. Lean Software Development. *The Project Perfect White Paper Collection* (www.projectperfect.com) consulté en février 2002.

[Langr 2002] Langr, J. Book Review: Questioning Extreme Programming. *2002. Site sur l'extreme Programming*

(http://www.xprogramming.com/xpmag/books20021009.htm) consulté en septembre 2005.

[Levine 2005] Levine, L. Reflections On Software Agility And Agile Methods: Challenges, Dilemmas,and the Way Ahead. Site du *Software Engineering Institute Carnegie Mellon University, Pittsburgh, PA U.S.A* (http://www.sei.cmu.edu/programs/acquisition-support/publications/reflections.pdf) consulté en août 2006.

[Martin 1992] Martin, J. *Rapid Application Development*. Prentice Hall,Upper Saddle River, New Jersey, 1992.

[Maslow 1943] Maslow, A.H. A Theory of Human Motivation. 1943. Sur le site de *Classics in the History of Psychology,* Originally Published in *Psychological Review,* 50, 370-396.

[Mcbreen 2002] McBreen, P. Pretending to Be Agile. 2002. Sur le site de *InformIT* (http://www.informit.com/articles/printerfriendly.asp?p=25913) consulté en juin 2006.

[Metaprog 2003] Pelrine, J., Davies, R. Support extreme programming with SCRUM. 2003. Sur le site de *Metaprog.com* (http://www.metaprog.com/) consulté en novembre 2005.

[Mumford 1983] Mumford, E. *Designing Human Systems, The ETHICS Method.* Sur le site de *Enid Mumford,* Livre en ligne (http://www.enid.u-net.com/C1book1.htm) consulté en mars 2004.

[NATO 1968] NATO Science commitee. Software Engineering. *proceeding of NATO Software Engineering Conference,* Garmish, Germany, 7th to 11th Oct 1968.

[Norton 2005] Norton, D. SCRUM Overview. 2005. Sur le site de CodeBetter.com (http://codebetter.com/blogs/darrell.norton/articles/50339.aspx) consulté en octobre 2005.

[OQLF 2006a] Office québécois de la langue française. Grand dictionnaire terminologique, Définition d'une norme. Disponible en ligne http://www.olf.gouv.qc.ca/ressources/gdt.html consulter en octobre 2006.

[OQLF 2006b] Office québécois de la langue française. Grand dictionnaire terminologique, Définition d'une norme. Disponible en ligne http://www.olf.gouv.qc.ca/ressources/gdt.html consulter en octobre 2006.

[Patton 2002] Patton, J. Hitting the Target: Adding Interaction Design to Agile Software Development. *Proceeding of SIGPLAN Notices 2002.*

[Pittman 1993] Pittman, M. Lessons Learned in Managing Object-Oriented Development. *IEEE Software,* Volume 10 , Issue 1, January, 1993, pp. 43-53.

[Poppendieck 2003a] Poppendieck, M. and Poppendieck, T. *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison-Wesley, 2003.

[Poppendieck 2003b] Poppendieck, M. Lean Software Development. *C++ Magazine, Methodology Issue,* Publication Fall 2003.

[Raymond 2003] Raymond, E.S. Discovering the obvious of Hacking and refactoring. 2003. Sur le site de *Artima Developper* (http://www.artima.com/weblogs/viewpost.jsp?thread=5342) consulté en avril 2006.

[Rosenhead 1998] Rosenhead, J. Complexity theory and management practice. 1998. Sur le site de *human-nature.com* (http://human-nature.com/science-as-culture/rosenhead.html) consulté en novembre 2005.

[Royce 1970] Royce, W.W. Managing the developement of large software systems. *Proceeding of IEEE WESON* August 1970, page 1-9.

[Schwaber 2001] K. Schwaber, Mike Beedle, *Agile Software Development with SCRUM.* Prentice Hall, 2001.

[Schwaber 2002] Schwaber, K. and Mar, K. Scrum with XP. 2002. Sur le site de *InformIT* (http://www.informit.com/articles/printerfriendly.asp?p=26057) consulté en octobre 2005.

[Schwaber 2004] Schwaber, K. *Agile Project Management with Scrum.* Microsoft Press, 2004.

[SEI 2005] Software Engineering Institute. Capability Maturity Model Integration (CMMi) overview. Sur le site officiel du *SEI, Carnegie Mellon* (http://www.sei.cmu.edu/cmmi/adoption/pdf/cmmi-overview05.pdf) consulté en novembre 2005.

[Stephens 2003] Stephens, M. and Rosenberg, D. *Extreme Programming Refactored: The Case Against XP.* Apress, August 5, 2003.

[Sutherland 2001] Sutherland, J. Inventing and Reinventing SCRUM in Five Companies. Sur le site officiel de l'*alliance agile* (http://www.agilealliance.org/system/article/file/888/file.pdf) consulté en novembre 2005.

[Sutherland 2003] Sutherland, J. SCRUM: Another way to think about scaling a project. Sur le site officiel de *Jeff Sutherland* (http://jeffsutherland.com/2003/03/scrum-another-way-to-think-about.html) consulté en avril 2006.

[Sutherland 2005] Sutherland, J. The Roots of Scrum: How Japanese Manufacturing Changed Global Software Development Practices. *Proceeding of JAOO,* Aarhus, Denmark, 28 Sep 2005.

[SWEBOK 2004] IEEE. *Guide to the Software Engineering Body of Knowledge, 2004 version SWEBOK (R).* IEEE Computer Society, Los Alamistos, California, 2004.

[Takeuchi 1986] Takeuchi, H. and Nonaka, I. The New New Product Development Game. *Harvard Business Rev.*, Jan./Feb. 1986, pp. 137-146.

[Vickoff 2002a] Vickoff, J.P. PUMA, Proposition pour l'Unification des Méthodes Agiles. Sur le site du *RAD Français* (http://www.rad.fr/forum02.pdf) consulté en mars 2006.

[Vickoff 2002b] Vickoff, J.P. PUMA, Proposition pour l'Unification des Méthodes Agiles. *La Lettre d'ADELI* n°48, Juillet 2002 p19-34.

[Vickoff 2002c] Vickoff, J.P. Le secours des méthodes "agiles". *L'informatique professionnelle* : 204 mai 2002, p. 12-16.

[VTT 2002] Abrahamsson, P. O. Salo, J. Ronkainen and J. Warsta. *Agile software development methods.* Sur le site de VTT Publication 478 (http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf) consulté en mai 2003.

[Wake 2002] Wake, W.C. *Extreme programming Explored.* Addison-Wesley, 2002.

[Wake 2004] Wake, W.C. Scrum Development Process. Sur le site de *XP123.com* (http://xp123.com/xplor/xp0507/Scrum-dev.pdf) consulté en octobre 2005.

[Wells 1999] Wells, D. Extreme Programming: A gentle introduction. Sur le site de *Extreme Programming* (http://www.extremeprogramming.org) consulté en septembre 2005.

[Wiki 2005] Merel, P. Description de Extreme Unified Process. Sur le site de *Cunningham and Cunningham Inc.* (http://xp.c2.com/ExtremeUnifiedProcess.html) consulté en avril 2006.

[Wikipedia 2006a] Encyclopédie libre en ligne Wikipédia. ISO 9001. Sur le site anglais de *Wikipedia.org* (http://fr.wikipedia.org/wiki/ISO_9001) consulté en août 2006.

[Wikipedia 2006e] Encyclopédie libre en ligne Wikipedia. Iterative and incremental development. Sur le site anglais de *Wikipedia.org* (http://en.wikipedia.org/wiki/Iterative_and_incremental_development) consulté en octobre 2006.

[Wikipedia 2006f] Encyclopédie libre en ligne Wikipédia. Cycle en V. Sur le site français de *Wikipédia.org* (http://fr.wikipedia.org/wiki/Cycle_en_V) consulté en septembre 2006.

[Wikipedia 2006g] Encyclopédie libre en ligne Wikipédia. 2TUP - Two Track Unified Process. Sur le site français de *Wikipédia.org* (http://fr.wikipedia.org/wiki/2TUP) consulté en juin 2007.

[Wikipedia 2006h] Encyclopédie libre en ligne Wikipedia. Delphi method. Sur le site anglais de *Wikipedia.org* (http://en.wikipedia.org/wiki/Delphi_method) consulté en juin 2007.

[Wikipedia 2007] Encyclopédie libre en ligne Wikipédia. Capability Maturity Model Integration. Sur le site français de *Wikipédia.org* (http://fr.wikipedia.org/wiki/CMMI) consulté en juin 2007.

[Windholtz 2006] Windholtz, M. Lean software development. Sur le site de *ObjectWind.com* (http://www.objectwind.com/papers/LeanSoftwareDevelopment.html) consulté en février 2006.

[Weinsten 2003] Weinsten, B. Eight reasons why extreme programming won’t work in your shop. 2003. *Sur le site de Builder.com (http://builder.com.com/5102-6315-1046490.html)* consulté en septembre 2005.

ANNEXES

Annexe A : Observations sur XP

Difficultés possibles

Extreme Programming (XP) peut aussi comporter certaines difficultés. En voici quelques-unes qui peuvent être rencontrées et comment elles peuvent être solutionnées [Beck 1999].

Sous estimer

Lorsque l'équipe réalise qu'elle est incapable de livrer autant de travail que ce qu'elle pensait, il faut tenter de voir si certaines pratiques n'ont pas été respectées. Si vous n'arrivez pas reprendre un rythme accéléré, il faut alors ré-estimer les scénarios en cours de réalisation, afin de réajuster l'ampleur du travail. Ensuite, il faut voir avec le client quels sont les scénarios les plus importants, afin de pouvoir laisser tomber les moins importants pour les remettre à plus tard.

Client non-coopérant

Lorsque nous avons un client qui ne s'implique pas dans la rédaction de tests et des scénarios, il est fortement possible que la relation de confiance ainsi qu'une bonne communication n'arrivent pas à s'établir. L'aide du client est requise, afin de lui expliquer et lui démontrer à quel point il est important pour la réussite du projet. Si personne ne se préoccupe du problème, c'est peut-être parce que le client ne juge pas le projet assez prioritaire pour aller de l'avant.

Roulement de personnel

Il arrive que les gens impliqués dans un projet quittent. Lorsqu'une personne quitte un projet XP, il ne part jamais avec un secret dont il est le seul à connaître. La programmation en pairs assure qu'au moins deux personnes connaissent chaque ligne de code. Complémentairement, l'automatisation des tests assure que les modifications apportées par

les autres membres de l'équipe ne causent aucun problème. Les nouvelles personnes dans l'équipe sont mises en pairs avec celles qui sont les plus expérimentées. Après quelques itérations, ils sont indiscernables des anciens membres de l'équipe. Par contre, un développeur qui ne s'intègre pas à l'équipe est aussi un problème. Parfois, les développeurs de haut calibre et ceux qui ne se débarrassent pas de leurs vieilles habitudes sont parfois nuisibles. Le projet peut souvent mieux se porter sans eux, peut importe leur niveau. Cette méthode préfère les développeurs sachant communiquer que l'excellence technique.

Changement de réquisitions

Pour XP, le changement n'est pas un problème. Le fait que le design minimaliste soit une pratique, aide à ce que le système puisse aller dans n'importe quelle direction par la suite. Il y a moins de choses à corriger qui ont été construites sur des spéculations. Si le client apporte un changement en cours d'itération, il est simplement mis sur la pile des scénarios à réaliser. Si ce changement est réellement urgent, il sera réalisé à l'itération suivante. La planification par itération ajoute un élément de simultanéité. Les choses étant planifiées et réalisées sur une courte échéance, le travail journalier est plus concret, car l'échelle de planification du projet est rapprochée.

Aspects critiqués de XP

Autant de visibilité entraîne plusieurs critiques. En 2003, Matt Stephens et Doug Rosenberg ont publié « Extreme Programming Refactored: The Case Against XP » [Stephens 2003], qui reproche principalement à XP que toutes ses pratiques sont interdépendantes et que peu d'organisations peuvent les implanter entièrement. Alors, le processus complet est un échec. En 2002, le livre « Questioning Extreme Programming » de Pete McBreed a été dédié à questionner XP [McBreen 2002]. Cet ouvrage analyse les pratiques et permet au lecteur de se questionner et de constater si cette méthode lui convient. Il questionne les coûts, les tests et les diverses pratiques. Il conclut que si une organisation pense à XP, c'est peut-être simplement qu'elle a besoin d'ajuster ses méthodes. C'est une bonne référence pour alimenter les débats sur la méthode [Glassborow 2002]. D'autres considèrent que plusieurs bénéfices sont mis de côtés et qu'il s'agit d'une

série d'arguments pour ne pas adopter XP [Langr 2002]. Kent Beck en a même écrit le préambule et exprime son désaccord.

Plus communément, XP est jugée complexe et minimaliste. Elle peut mener à la confusion, à l'ambiguïté, à l'oubli de spécifications et à de mauvais design. On dénote son manque d'architecture globale préalable au développement. L'utilisation de la programmation en pairs est démesurée pour des problèmes peu complexes. L'omniprésence du client diminue la production de documents, ce qui laisse peu de trace. Il peut être très sollicité, ce qui peut bloquer ou compromettre le projet. La discipline qu'exige cette méthode la rend fragile et risquée. [Stephens 2003; Weinsten 2003].

XP tel un oignon

Dans l'ouvrage de William C. Wake [Wake 2002], on présente les pratiques d'XP regroupées selon trois niveaux. Avec de tels ensembles, il est possible d'imaginer une implantation progressive pour les équipes qui désirent suivre cette méthode. Les développeurs peuvent être invités à suivre les pratiques individuelles en premier. Ensuite, celles d'équipes et pour finir celles de processus. De cette manière, la transition vers XP sera moins imposante, car cette méthode est rigoureuse et demande beaucoup de discipline aux équipes.

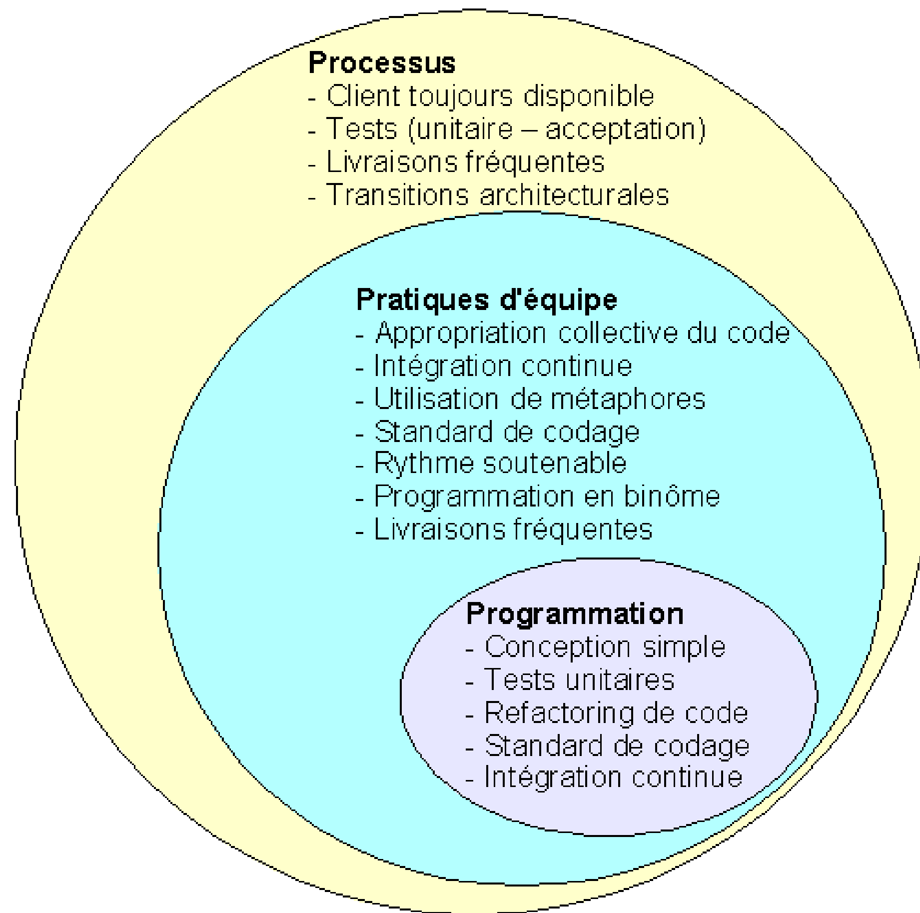


Figure 5-19 : XP vue en couche d'oignon

Au centre, nous avons les pratiques (disciplines) de programmations individuelles que chaque développeur peut mettre en pratique de manière autonome. Comme il est difficile de changer ses habitudes, l’autodiscipline commence ici. La couche médiane concerne les pratiques d’équipe. Lorsqu’un développeur réalise l’importance de suivre ses pratiques, le partage du code et le temps requis pour le modifier est amélioré. Autour, nous avons les pratiques de gestion de projet, qui fournissent les indicateurs d’avancement et de planification. Il existe peu de métriques de suivi de projets avec XP, c’est pourquoi le nombre de fonctionnalités réussissant leur test unitaire est très important. Nous pouvons constater que quelques pratiques se répètent, car elles rejoignent plusieurs catégories.

Annexe B : Observations sur SCRUM

Bien que les méthodes agiles ne se targuent pas de résoudre tous les problèmes, nous avons souvent tendance à les soumettre aux paradigmes régissant les pratiques courantes en gestion de projets. Ce qui amène souvent des questions qui tentent d'étirer leur cadre d'utilisation. Face aux différentes problématiques soulevées, une série de propositions est amenée, afin de proposer des pistes de solution à ces questionnements.

Projet de grande envergure

Une des premières observations qui survient dans les projets agiles, c'est qu'ils sont limités à des petits projets impliquant peu de personnel. SCRUM s'adresse principalement aux équipes de moins de 10 personnes. Il est possible d'envisager une adaptation pour de plus gros projet. La stratégie consiste à construire une structure de plus haut niveau, ce qui équivaut à synchroniser plusieurs équipes de moins de 10 personnes [ADM 2003; ADM 1996a].

Une alternative proposée consiste à subdiviser le projet en plusieurs petits projets. Le backlog du produit est ainsi découpé en sous projet. Le Scrummaster peut alors rencontrer subséquemment les différentes équipes s'occupant chacune d'un backlog de sprint. Plusieurs sprints sont réalisés parallèlement, suivi par un seul Scrummaster qui permet l'harmonisation de toutes les équipes [ADM 2003].

Une autre alternative consiste à bâtir un Métha-Scrum, qui constitue une structure pyramidale. Le backlog produit est subdivisé en sous projets pour dresser les backlog de sprint. Avec cette approche, la participation de plusieurs « Scrummasters » est requise. Au départ, un « scrum » de premier niveau est composé de quelques « Scrummasters ». Ceux-ci conduiront ensuite des « scrums » intermédiaires, réunissant que des « Scrummasters ». Finalement, ces derniers rencontrent leurs équipes respectives avec des « scrums » de troisième niveau. Ce modèle pourrait permettre l'implication de plus de 200 développeurs [Sutherland 2005].

Les contrats forfaitaires

Une autre constatation envers SCRUM et l'ensemble des approches agiles, c'est leur difficulté à pouvoir proposer des contrats forfaitaires de durée et coûts fixes. Sans prétendre être plus précis que les méthodes classiques, une approche est tout de même proposée. Dans ce cas, une vision plus précise doit être construite. Les changements seront traités selon leur importance, en échangeant avec des éléments planifiés de valeurs équivalentes. La problématique du contrat forfaitaire soulève des interrogations. Voici une approche pour solutionner le problème [ADM 2003].

- Développer une vision des éléments importants, selon la perspective du produit.
- Créer le backlog, produire des éléments fonctionnels et non-fonctionnels.
- Réviser, prioriser et évaluer les éléments avec le client, à la lumière de la vision du produit.
- Créer une architecture et une conception pour mieux évaluer. Mettre l'emphase sur les éléments qui maximisent la valeur d'affaire.
- Discuter avec le client comment les éléments seront livrés, en lui rappelant qu'il est libre de changer les priorités et les éléments tant que l'estimation reste similaire.
- Soumettre l'offre basée sur le backlog du produit.

De plus, cette approche laisse un peu de latitude sur les événements pouvant survenir en cours de projet, sans avoir à renégocier.

- Tout élément n'étant pas commencé, peut être remplacé par un autre de valeur équivalente.
- Les priorités et les éléments peuvent être changés.
- Le client peut demander des versions additionnelles, à tout moment, selon les conditions établies.
- Le client peut mettre un terme au contrat rapidement, si la valeur du produit est satisfaisante, sans compléter la balance non-facturée.

Finalement, il faut parfois choisir quelle variable aura priorité : Prix fixe, temps fixe, date repoussée ou dépassement des coûts.

Niveau de maturité CMMi

Le modèle de maturité CMMi (annexe D) du SEI [SEI 2005] est souvent cité en exemple comme un idéal à atteindre. Bien qu'à l'origine il ait été conçu dans les années 1980, pour qualifier les fournisseurs du département de la défense américaine (DoD), ce modèle est

demeuré une référence [Clever 2004]. La firme ADM doute de l'efficacité d'une telle philosophie [ADM 2005-2] mais, s'adonne tout de même à estimer le niveau de SCRUM sur leur échelle. (Voir Annexe Modèle du CMMi) Pour sa part, SCRUM peut couvrir suffisamment de pratiques du modèle pour se classer au niveau 3 du CMMi [Anderson 2005; ADM 2003]. Avec les éléments exigés, SCRUM peut atteindre ce niveau sans compromettre son agilité.

Valeur d'affaire du système

Fondamentalement, le produit développé justifie son investissement par les bénéfices qu'il apporte à l'entreprise. La valeur d'affaire pouvant se traduire par des économies ou autres types d'avantages, le calcul du retour sur investissement (RoI) est une donnée importante qui justifie les montants impliqués. L'approche incrémentale gérée par priorités, apporte rapidement un bénéfice tangible à l'entreprise, le RoI est plus facilement démontrable. Il est possible de suivre un modèle de réinvestissement des bénéfices, qui permet d'amortir plus facilement les coûts du projet. De plus, en bénéficiant graduellement de fonctionnalités opérationnelles, l'entreprise diminue le temps requis pour tirer avantage du produit [Cohn 2005b].

Le système essentiel

Une telle approche entraîne un effet de contingentement, qui limite les envolés d'efforts incontrôlables. Entre la vision idéalisée du produit au départ et l'ensemble des éléments qui seront ajustés au cours de son développement, émerge le système essentiel. Tout élément superflu engendre des coûts et un entretien supplémentaire. Le système essentiel est le résultat des éléments, apportant une valeur d'affaire qui fourni un avantage en fonction des montants investis, pour une qualité acceptable de produit [ADM 2005].

Complémentarité avec XP

SCRUM est au suivi de projet, ce que XP est à la programmation. SCRUM insiste plus sur la gestion et le contrôle du projet et ne suggère aucune pratique de programmation laissant cette section dans la boîte noire. XP peut facilement y être intégré pour combler cet aspect

[Sutherland 2001; Schwaber 2002; ADM 2003; Metaprog 2003]. La proposition est d'ajouter à SCRUM les pratiques de programmation comprises dans XP. Il en résultera un processus plus complet, mais demandant aussi plus de discipline [Schwaber 2002]. Mike Beedle nomma cette initiative Xbreed, mais ne semble pas avoir percé.

Augmentation de la productivité

On note généralement une augmentation de productivité de 10 à 20%, en comparant l'utilisation de SCRUM et un projet similaire suivant les standards de l'industrie. À titre d'exemple, certaines métriques ont été extraite du projet de Borland Quattro pour Windows. Ces métriques illustrent une productivité largement supérieure à celle de l'industrie [Sutherland 2003].

Avantages et contraintes

Cette approche est avantageuse pour la gestion de projet, le travail d'équipe et la satisfaction du client. Elle permet de gérer les améliorations du produit, par morceaux et dans un environnement changeant. Elle permet une meilleure communication et motive l'équipe. Le client devient un partenaire dans le succès de la réalisation de son produit.

Par contre, elle demande plus de rigueur de la part du gestionnaire, qui doit favoriser un suivi qualitatif et suivre continuellement la progression du projet. Le gestionnaire doit autoriser l'équipe à prendre des décisions. Cette nouvelle approche peut se voir freiner par la résistance au changement et indisposer par la responsabilité qu'elle incombe aux gens [Norton 2005; BI 2001; IEEE 2000].

Synthèse de SCRUM

Pour terminer cette annexe, voici une excellente synthèse de SCRUM, présentée en une page par Wake.

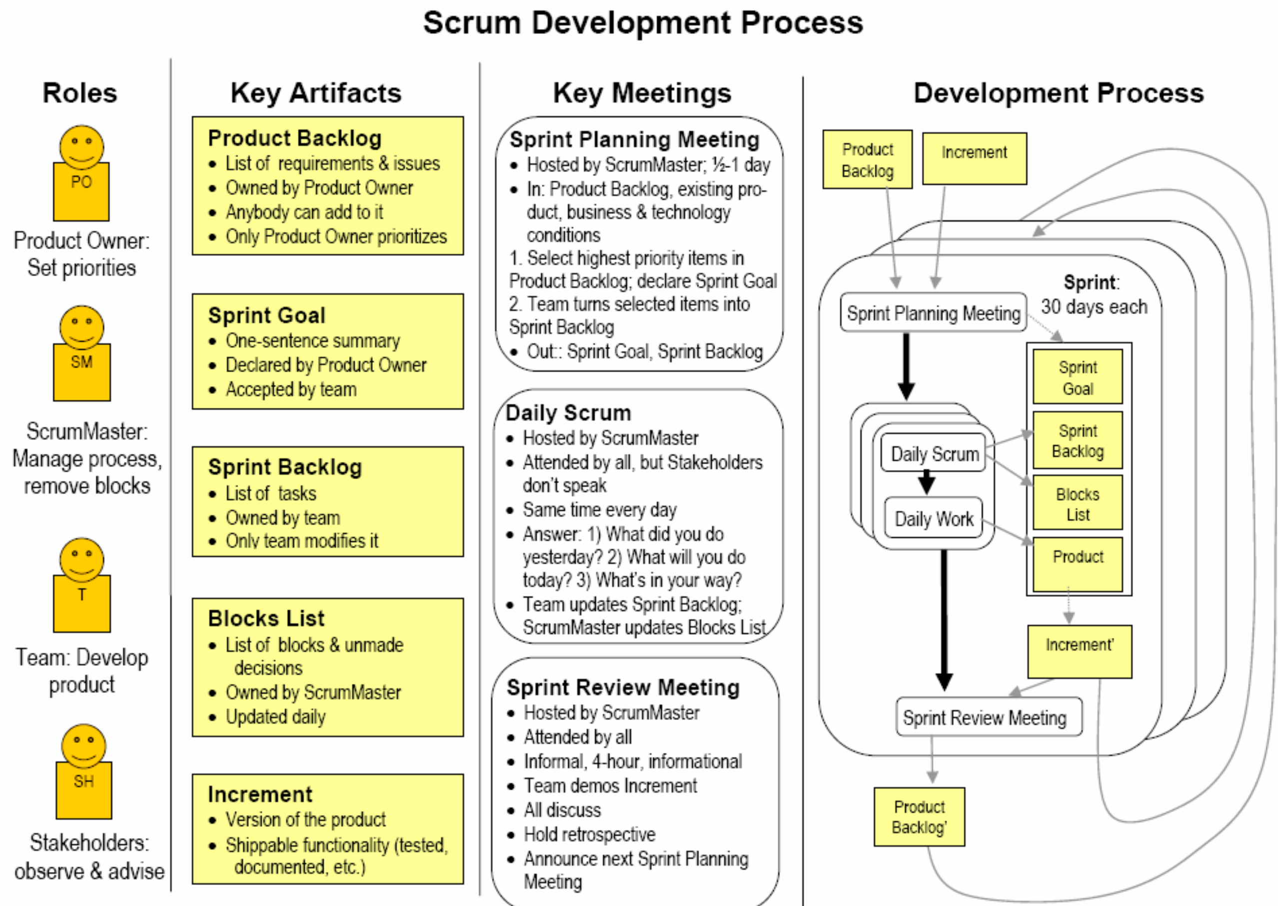


Figure 5-20 : SCRUM en une page de Wake

[Tiré de Wake 2004].

Annexe C : Observations sur Crystal Clear

Il existe peu de critique spécifique sur Crystal, étant donné son rayonnement limité. Son inventeur a tout de même jugé bon de clarifier certains aspects. De plus, Crystal étant souvent similaire à SCRUM, plusieurs critiques peuvent lui être transférés. Dépendamment de ce qui est priorisé, une méthode prend une orientation : soit de répétitivité comme ISO-9000, de prévisibilité et de contrôle comme CMMi ou de productivité comme XP. Cette section positionne Crystal dans cet éventail [Cockburn 2004].

Crystal et XP

XP priorise la productivité et l'évolution du code, il réduit les travaux intermédiaires et raccourci le cycle des itérations à deux semaines. Cette méthode partage plusieurs pratiques avec Crystal, mais demeure plus stricte sur l'adoption de chacune de ses pratiques.

Toutes deux favorisent la communication, le travail d'équipe et la programmation en duo. Les deux méthodes proposent une approche minimaliste sur la production du code, afin d'augmenter la vélocité et réduire les ajustements. L'environnement technique est aussi très similaire.

De son côté, Crystal demeure plus souple sur les pratiques. L'équipe décide des pratiques qu'elle conserve. Par contre, Crystal est plus rigoureux sur le travail préalable et la planification des étapes subséquentes. La production de la documentation est aussi un point important pour Crystal, ce que l'on ne retrouve pas chez XP.

Il est possible pour une équipe Crystal de suivre les pratique de XP. Si vous désirez suivre XP au complet et à la fois suivre Crystal, vous devrez alors livrer le produit à l'utilisateur plus souvent et investir plus d'effort sur la documentation.

Crystal et SCRUM

SCRUM s'appuie sur trois pratiques clefs. Ces pratiques sont : des itérations fixes dans le temps, une liste de demandes dynamiques et des rencontres journalières. Ces trois éléments trouvent leur équivalent dans Crystal.

La seconde caractéristique importante de SCRUM, est que cette méthode ne propose pas de pratique de réalisation précise. Cette « boîte noire » laisse toute la place pour introduire les pratiques judicieuses. Sur ce point, Crystal comporte un ensemble de départ qui aide les équipes à combler ce vide et analyser leurs besoins méthodologiques. Avec cet ensemble de pratiques et d'outils proposés, Crystal aide à rapprocher rapidement les membres de l'équipes.

Crystal et RUP

Rational Unified Process (RUP) peut être adapté au même point que Crystal peut l'être. Les deux méthodes sont des « générateurs » de méthodes. Ce qui distingue les deux approches, est que Crystal part d'un ensemble restreint et suggèrent d'ajouter des éléments aux besoins. L'équipe peut partir d'où elle est, afin d'ajouter et d'ajuster sa méthode. Crystal demande peu au départ. Mis à part ses livres, Crystal n'a aucune visée commerciale.

Pour RUP, l'ensemble de départ est complet et la méthode suggère de se limiter à ce qui n'est pas nécessaire. Le noyau de RUP réside sur l'architecture, la modélisation visuelle et son outil de modélisation UML. Il est possible que les deux approches mènent les équipes à suivre les mêmes conventions. Elles partagent plusieurs pratiques équivalentes. Malgré les diminutions que peut subir RUP, cette méthode exige tout de même de construire les modèles préalablement, ce qui n'est pas une nécessité pour Crystal.

Crystal n'est pas basé sur un outil complexe ou un formalisme standard. Il laisse la liberté aux équipes d'y adhérer. De cette manière, des solutions novatrices peuvent émerger de l'équipe et correspondre plus adéquatement à son besoin. Un des aspects qui facilite l'interaction avec les clients, est l'utilisation d'outils et le formalisme simple, ce qui leur permet d'être compris. Il est possible pour un projet RUP de ressembler à un projet agile, sans nécessairement le ressentir.

Crystal et le CMMi

À la base, Crystal n'est pas conçu pour être certifié par le CMMi. En s'efforçant d'ajouter certaines pratiques qui permettent de rencontrer ses exigences, nous pouvons envisager une certification de niveau 3 (tout comme SCRUM). Par contre, l'expérience a fait ressortir quelques problèmes d'adaptation en ce qui a trait à la proximité des clients et la qualité de la communication de l'équipe. De plus, la définition par écrit du processus laisse peu de place à sa rectification. Les équipes qui réussissent à rédiger leur processus, s'imaginent qu'il est figé et ne peut évoluer.

Pour Crystal, le processus de production logiciel est empirique et ne peut être défini que dans ces grandes lignes. Les équipes révisent constamment leur processus au début et en cours de projet, ce qui le rend difficile à rédiger.

Crystal inverse la pyramide du CMMi (voir l'annexe E : Modèle du CMMi). Le niveau 5 demande d'optimiser son processus en fonction des métriques du niveau 4. L'un des principes de Crystal, est de s'analyser pour optimiser le travail. Sans exiger de mettre en place une série de métriques, elle demande de suivre un sens commun sur les pratiques qui optimisent les efforts, tout en étant viable.

Tableau de classification des métriques agiles

Une équipe qui désire connaître son niveau d'agilité ne peut s'évaluer facilement. Il n'existe pas d'échelle de qualification. À défaut d'un outil formel, Alistair Cockburn propose une série de mesures et de questions qui permettent de déterminer la position de votre contexte organisationnel [Cockburn 2005].

Mesure	Valeur souhaitable
Nombre de personnes sur le projet	< 30
Longueur des itérations	< 2 mois
Fréquence des livraisons au client	< 3 mois
Fréquence des présentations au client	< 1 mois
Fréquence des ateliers de réflexion	< 1 mois
Communication osmotique	Oui
Sécurité personnelle	Oui
Concentration (temps et priorité consacré)	Oui
Accès facile à l'utilisateur expert	Oui
Gestion de la configuration	Oui
Automatisation des tests	Oui
Fréquence des intégrations	Continue
Collaboration des autres secteurs de l'organisation	Oui
Exploration 360°	Oui
Victoire rapide	Oui
Squelette marchant	Oui
Ré-architecture incrémentale	Oui
Radiateurs d'information	Oui
Programmation en pairs	Possiblement
Programmation côte-à-côte	Possiblement
Développement dirigé par les tests	Possiblement
Blitz de planification	Oui
Rencontre journalière	Oui
Tableau de progression	Oui
Liste de priorité dynamique	Possiblement

Tableau 5-10 : Métriques agiles de Crystal

[Tiré de Cockburn 2005]

Annexe D : Autres approches complémentaires

Il existe plusieurs autres approches qui gravitent autour des méthodes agiles. Le site de l'alliance agile répertorie les diverses méthodes reconnues et aussi des approches complémentaires [AA 2005b]. Cet ensemble est qualifié d'écosystème.

Le Canadien Scott W Ambler, propose une série d'approches complémentaires à l'utilisation d'une méthode agile. Ces compléments touchent différentes facettes du développement [Ambler 2005a].

- Modélisation Agile (Agile Modeling ou AM) se veut un ajout aidant à la conception du système. Elle focalise sur la modélisation et sa documentation. Elle se greffe à une méthode plus globale telle que SCRUM ou XP.
- Technique de base de données agile (Agile Database Technique ou AD) est une technique complémentaire proposée pour les bases de données.
- Design mené par les Tests (Test Driven Development ou TDD) est aussi une pratique ciblée qui s'insère au niveau de la réalisation. Elle demande l'utilisation d'outils d'automatisations des tests, elle préconise le développement de tests unitaires tout au long du processus de réalisation.

Le processus unifié (Unified Process ou UP) est un gabarit de méthode qui supporte les différents diagrammes UML. UP est souvent reconnu comme une version plus moderne des approches traditionnelles, car l'ensemble des éléments de documentation exige qu'un processus fortement documenté s'y retrouve. Son application est parfois jugée lourde et complexe [Hesse 2003]. Pris dans sa forme originale, elle ne se classe pas parmi les méthodes agiles. Elle est surtout dirigée par ces modèles et laisse peu de place aux changements durant le projet [BI 2001]. Par contre, comme le spécifie UP, une adaptation de la méthode en fonction du projet auquel elle s'applique est conseillée. Cette adaptation peut être agile.

À partir de ce cadre, une série d'instance ont été construite.

- Rational Unified Process (RUP) est l'adaptation de Rational Software (maintenant IBM) du gabarit du processus unifié. Elle est aussi l'instance la plus connue [IBM 2006].
- Enterprise Unified Process (EUP) est une adaptation intégrant les phases de post-implantations et décrivant le cycle de vie du logiciel [Ambler 2005b].
- Extreme Unified Process (XUP) est une adaptation hybride, intégrant UP avec les pratiques d'extreme Programming [Wiki 2005].
- Agile Unified Process (AUP) est une adaptation permettant l'agilité du développement, cette méthode met l'accent sur l'optimisation et l'efficience sur le terrain, plus que sur le modèle théorique [Ambler 2005c].
- Two track UP (2TUP) est une adaptation partant de deux voies différentes. Une cueillette des besoins fonctionnels et techniques. Les deux sources joignant le design du modèle objet. Le cycle de développement s'apparente à un Y [Wikipedia 2006g].

Le modèle en spirale gagnant- gagnant (Win-Win Spiral) proposé par Barry Boehm, est une mise à jour de son populaire modèle en spirale. Elle ajoute l'implication de tous les intervenants. De cette collaboration, en découle une priorisation des spécifications. En ajoutant ces pratiques de feedback récurent, le développement est écourté et de meilleure qualité. De cette manière, tous les intervenants sont gagnants [Boehm 1998].

Développement Rapide d'Application (Rapid Application Development ou RAD) de James Martin est un gabarit de méthode, basé sur la construction de prototypes présentés au client [Martin 1992]. Étant donné sa nature de gabarit, elle ne se classe pas comme une méthode agile. Certains mettent l'emphase sur son aspect traditionnel. Les prototypes remplacent en majeur partie les spécifications lorsque le client approuve ceux-ci, le processus s'enclenche de façon linéaire, laissant peu de place aux changements. C'est pourquoi elle ne se qualifie pas complètement dans les méthodes agiles [BI 2001]. D'autres détracteurs les classent simplement parmi elles [Vickoff 2002b]. Il demeure tout de même une ambiguïté au sujet de ce que l'on considère comme un prototype. Est-ce un code réduit de production ou est-ce un code qui sera refait? Quel qu'en soit le qualificatif du RAD, elle est tout de même à l'origine de deux méthodes agiles reconnues : DSDM et ASD.

En 2002, Vickoff publia une Proposition pour Unifier des Méthodes Agiles (PUMA) [Vickoff 2002a]. Cette proposition analysait le points communs, pour en faire ressortir les éléments communs. Il ne semble pas avoir eu de suite à cette idée. Cette proposition n'allait pas dans le même sens que les Agilistes fondateurs du manifeste. Dès leur première rencontre, ils avaient écarté la possibilité de fusionner leurs approches [Fowler 2001]. Le logiciel libre (Open Source ou OS) est parfois classé agile, car les quatre critères définissant l'agilité sont rencontrés. En s'appuyant sur son potentiel pour livrer des logiciels fonctionnels, s'adaptant au changement issu de l'expérience de ses utilisateurs, nous pouvons effectivement les classer parmi elles. Par contre, ce style de développement est souvent géré par des responsables de produit géographiquement distribués, impliquant différents développeurs qui ne se rencontreront probablement jamais. C'est une forme de collaboration valable, mais l'agilité est utilisée pour créer une synergie avec des équipes travaillant ensemble. Elle augmente la confiance avec le client, afin de mettre au second plan la négociation de contrat souvent issue des changements au projet. Ce type d'approche n'est pas soumis aux pressions commerciales, leurs ressources et leurs durées ne sont pas limitées. C'est pourquoi elle n'est pas définie agile [Cockburn 2001]. Éric Raymond s'accorde avec Fowler pour soutenir que l'Open Source est plus un style qu'une méthode [Raymond 2003].

Finalement, une autre approche peu documentée originaire d'Angleterre nommée « Effective Technical and Human Implementation of Computer-based Systems (ETHICS) », est une méthode élaborée dans les années 1980 par Enid Mumford [Mumford 1983]. Cette méthode peu connue, met l'utilisateur au centre du processus. Elle considère l'aspect sociotechnique et prend en compte plusieurs facteurs humains. Elle couvre l'aspect de l'apprentissage du changement, de la collaboration et de la satisfaction au travail. Par contre, cette approche reste évasive sur la manière de réaliser le produit, ce qui ne permet pas de classifier parmi les méthodes agiles. On peut tout de même souligner son existence et surtout sa préoccupation pour les valeurs sociales entourant les techniques informatiques.

Annexe E : Modèle du CMMi

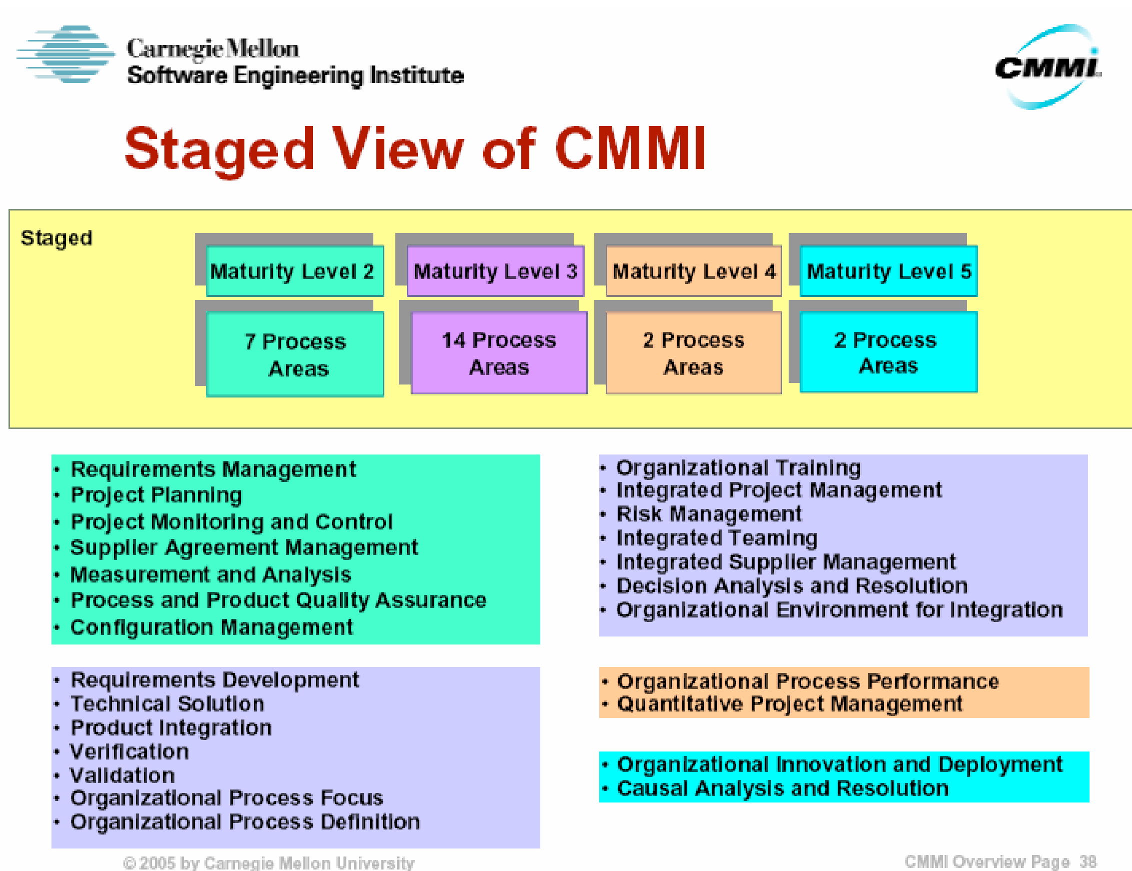


Figure 5-21 : Modèle des niveaux du CMMi.

[Tiré de SEI 2005]

Annexe F : Les racines de XP

Voici un extrait [Beck 1999] qui liste les différentes sources d'inspirations de XP.

Roots of XP

The individual practices in XP are not by any means new. Many people have come to similar conclusions about the best way to deliver software in environments where requirements change violently.[1-3]

The strict split between business and technical decision making in XP comes from the work of the architect Christopher Alexander, in particular his work *The Timeless Way of Building*,[4] where he says that the people who occupy a structure should (in conjunction with a building professional) be the ones to make the high-impact decisions about it.

XP's rapid evolution of a plan in response to business or technical changes echoes the Scrum methodology[5] and Ward Cunningham's Episodes pattern language.[6]

The emphasis on specifying and scheduling projects from the perspective of features comes from Ivar Jacobson's work on use cases.[7]

Tom Gilb is the guru of evolutionary delivery. His recent writings on EVO[8] focus on getting the software into production in a matter of weeks, then growing it from there.

Barry Boehm's Spiral Model was the initial response to the waterfall.[9] Dave Thomas and his colleagues at Object Technology International have long been champions of exploiting powerful technology with their JIT method.[10]

XP's use of metaphors comes from George Lakoff and Mark Johnson's books, the latest of which is *Philosophy in the Flesh*.[11] It also comes from Richard Coyne, who links metaphor with software development from the perspective of postmodern philosophy.[12]

Finally, XP's attitude toward the effects of office space on programmers comes from Jim Coplien,[13] Tom DeMarco, and Tim Lister,[14] who talk about the importance of the physical environment on programmers.

References

1. J. Wood and D. Silver, *Joint Application Development*, John Wiley & Sons, New York, 1995.
2. J. Martin, *Rapid Application Development*, Prentice Hall, Upper Saddle River, N.J., 1992.
3. J. Stapleton, *Dynamic Systems Development Method*, Addison Wesley Longman, Reading, Mass., 1997.
4. C. Alexander, *The Timeless Way of Building*, Oxford University Press, New York, 1979.
5. H. Takeuchi and I. Nonaka, "The New Product Development Game," *Harvard Business Rev.*, Jan./Feb. 1986, pp. 137-146.
6. W. Cunningham, "Episodes: A Pattern Language of Competitive Development," *Pattern Languages of Program Design 2*, J. Vlissides, ed., Addison-Wesley, New York, 1996.
7. I. Jacobsen, *Object-Oriented Software Engineering*, Addison-Wesley, New York, 1994.
8. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Wokingham, UK, 1988.
9. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
10. D. Thomas, "Web Time Software Development," *Software Development*, Oct. 1998, p. 80.
11. G. Lakoff and M. Johnson, *Philosophy in the Flesh*, Basic Books, New York, 1998.
12. R. Coyne, *Designing Information Technology in the Postmodern Age*, MIT Press, Cambridge, Mass., 1995.
13. J.O. Coplien, "A Generative Development Process Pattern Language," *The Patterns Handbook*, L. Rising, ed., Cambridge University Press, New York, 1998, pp. 243-300.
14. T. DeMarco and T. Lister, *Peopleware*, Dorset House, New York, 1999.