REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE M'HAMED BOUGARA – BOUMERDES
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUEE

Résumé de Mémoire de Magistère

THEME

Une Approche pour la Sélection de Méthodologies de Développement Logiciel

Présenté par

M^{r} TOUIL Ghassen

Soutenu publiquement le ……

Membres du jury:

M. Mezghiche	(Professeur U.M.B.B.)	Président
M. Ahmed-Nacer	(Professeur U.S.T.H.B.)	Rapporteur
Z. Alimazighi	(Professeur U.S.T.H.B.)	Examinateur
A. Harzallah	(Docteur U.M.B.B.)	Examinateur
R. Ahmed-Ouamer	(Maître de conférence U.M.M.T.O.)	Examinateur

ANNEE -2006-

ملـــــــــــخص

تعتبر العملية البرمجية Software Process من أحدث الطرق المستعملة لمعالجة الطبيعة المعقدة للبرامج. و قد عرفت العمليات البرمجية انتشارا و تنوعا كبيرين في الفترة الأخيرة. هذا التنوع ناتج عن الاختلاف في المبادئ الأساسية لهذه العمليات البرمجية، فكان من نتائج هذا التنوع تحديد مجال استخدام كل عملية برمجية في عدد معين من الميادين. من جهة أخرى يعتبر اختيار العملية البرمجية المناسبة عنصرا حاسما لتحديد مدى نجاح أي مشروع تطوير برامج. و نظرا للعدد الكبير للعمليات البرمجية المتوفرة حاليا و كذلك الاهتمامات المختلفة التي يمليها علينا مشروع تطوير برامج معين يجعلان موافقة المشروع بالعملية البرمجية الملائمة أمرا غاية في الصعوبة.

في دراستنا هذه نقدم بعض التوجيهات لمساعدة الشركات لاتخاذ القرارات المناسبة فيما يخص اختيار العملية البرمجية المناسبة لتطوير برنامج ما. و قد قدمنا مجموعة من العوامل المتعلقة بالعناصر الثلاثة التالية : المشروع، الشركة، و العملية البرمجية، واستعملناها للمقارنة بين المنهجيات التالية : Rational Unified Process, Extreme Programming, Cleanroom Software Engineering, Open Source Development. العوامل المستعملة تضم أهم العناصر المعينة على اختيار العملية البرمجية. وفي الأخير استعملنا بطاقة الأداء المتوازن Balanced Scorecard لتوجيه عملية الاختيا،ر و تم شرح استخدام هذه الطريقة عبر دراسة حالة لاختيار المنهجية العملية المناسبة.

ABSTRACT

Software processes or methodologies are a recent approach to address the increasing complexity of software. In recent years, we witnessed a growth and diversity in software development methodologies. Underlying principles make software development methodologies different and define a range of software projects that can be dealt with. Selecting the correct methodology is a critical factor to the success of any project. The large number of software methodologies available today and the different concerns that a project may arise make it difficult to match the project with the suitable methodology. In the present study we provide some guidelines that assist organizations to make decisions about the methodology to be used for developing a given product. A framework of factors in relation with methodology, project, and organization was provided and applied to compare the following four methodologies, which represent the mainstream in software development processes: Rational Unified Process, Extreme Programming, Cleanroom Software Engineering and Open Source Development. The framework includes the main driving factors when selecting a methodology. The Balanced Scorecard model with its four complementary perspectives was used to guide the selection process. The application of such a model was presented and illustrated in a case study for selecting a project methodology.

KEYWORDS: Software Development Processes, Software Process Comparison, Software Process Selection, Rational Unified Process, Cleanroom Software Engineering, Extreme Programming, Open Source Development, Balanced Scorecard.

Une Approche pour la Sélection de Méthodologies de Développement Logiciel

ABSTRACT

Les processus ou les méthodologies de développement logiciel sont une approche récente pour adresser la complexité croissante du logiciel. Ces dernières années, nous avons vécu une croissance et une diversité de méthodologies de développement logiciel. Les principes de base rendent ces méthodologies de développement différentes et définissent une gamme de projets de logiciel qui peuvent être traités. Le choix de la méthodologie correcte est un facteur critique pour le succès de n'importe quel projet. Le grand nombre de méthodologies de développement logiciel disponibles aujourd'hui et les différents soucis qu'un projet peut faire sortir rendent difficile de concorder un projet avec une méthodologie appropriée. Dans la présente étude nous fournissons quelques directives qui aideront les organisations à prendre des décisions sur la méthodologie à employer pour développer un produit donné. Un framework de facteurs en relation avec la méthodologie, le projet et l'organisation a été fourni et appliqué pour comparer les quatre méthodologies suivantes, qui représentent le courant principal dans le développement logiciel : Rational Unified Process, Extreme Programming, Cleanroom Software Engineering et Open Source Development. Le framework inclut les facteurs les plus important lors de la sélection d'une méthodologie. Le Balanced Scorecard avec ses quatre perspectives complémentaires a été employé pour guider le procédé de sélection. L'application d'un tel modèle a été présentée et illustrée dans une étude de cas pour choisir une méthodologie de développement pour un projet.

MOTS-CLÉS: Processus de développement logiciel, Comparaison de processus de développement logiciel, Sélection de processus de développement logiciel, Rational Unified Process, Extreme Programming, Cleanroom Software Engineering, Open Source Développement, Balanced Scorecard.

I. Introduction Générale

I.1. Introduction

Le processus de développement logiciel est devenu un souci central pour la communauté du génie logiciel. Étant un successeur des modèles de cycle de vie (modèle à chute d'eau, spirale, à incréments, etc.), le processus de développement logiciel fournit plus de directives sur la façon de développer un logiciel. Beaucoup de praticiens ont consacré leurs efforts dans la recherche d'un *"processus unifié"* ou *"une méthodologie unifiée"* qui conviendra à développer n'importe quel projet, imitant ainsi le travail fait pour unifier les langages de modélisation en UML, le standard industriel pour la modélisation.

La réponse à cette issue a été fournie dans le célèbre article de Frederick Brooks : *"No Silver Bullet"* [1]. L'auteur a réclamé qu'il n'y aura aucune méthode de développement qui répondra aux exigences excessives des développeurs et des clients. Cette idée a été confirmée plus tard par des praticiens du logiciel lorsqu'une douzaine de méthodes à différents niveaux de contraste sont apparues, en réponse à la diversité du logiciel.

Les tentatives d'unifier ces méthodes ont eu comme conséquence plus de diversité et des familles de méthodologies de développement logiciel sont ainsi apparues au lieu de la méthodologie unifiée promise. Ceci a été reflété principalement dans les trois projets suivants:

- Le Processus Unifié, le prédécesseur du Rational Unified Process, résultant de la fusion de trois méthodes orientés-objet : Booch, OMT et OOSE.
- Le développement agile, lancé par *"The agile manifesto"*, qui regroupe un ensemble de méthodologies de développement logiciel mettant l'être humain au centre du développement.
- Open Process Framework, soutenu par *"OPEN Consortium"*, basé sur la contribution de plusieurs méthodes de développement: SOMA, FÈVE, Syntropy, etc.

D'autres méthodologies isolées sont apparues avec des soucis ou des besoins spéciaux telles que : Cleanroom, Open Source Development, Reverse Engineering Development et Offshore Development Methodology.

Afin de rendre l'exécution quotidienne des projets plus facile, les chefs de projets devraient réfléchir sur la méthodologie à employer et décider sur la méthodologie la plus adéquate. Adopter une méthodologie incorrecte peut causer des dépassements de budgets et de délais aussi bien qu'une qualité inférieure pour le produit. Les priorités du projet influenceront considérablement le choix des chefs de projets. Ces derniers peuvent se concentrer sur la qualité, la productivité, la livraison rapide ou même l'apprentissage.

Vu le grand nombre de méthodologies disponibles aujourd'hui, de nombreux chercheurs du génie logiciel se sont intéressés à comparer ces méthodologies. En étudiant et en comparant ces dernières, nous pouvons comprendre le contexte dans lequel une méthodologie pourrait être appliquée et donnerait plus de valeur. Une telle comparaison peut aider les organisations et les chefs de projets dans leur choix de la bonne méthodologie pour chaque projet.

I.2. Objectifs et contributions de cette étude

La diversité des domaines d'application du logiciel et des priorités des chefs de projets a eu comme conséquence différentes approches de développement logiciel. Ces approches ou méthodologies exigent différents types de ressources et donnent différents genres de produits. Comparer ces méthodologies fait apparaître les différences et les aspects contradictoires entre elles et permet ainsi de découvrir leurs points forts et faibles.

Dans notre étude, nous traitons les méthodologies du point de vue utilisateur. L'utilisateur est représenté par le chef de projet et les membres de l'équipe de développement, voulant obtenir le maximum de la méthodologie utilisée. Quatre des méthodologies les plus utilisées dans le monde du développement logiciel ont été choisies pour entreprendre cette étude : Rational Unified Process, Cleanroom Software Engineering, Extreme Programming et Open Source Development. Travailler avec plus de méthodologies demanderait une étude plus approfondie et plus de temps, ce qui n'a été pas fourni pour cette étude.

Notre but est d'évaluer les différents aspects des quatre méthodologies et de fournir une analyse comparative qui peut servir de framework pour choisir les méthodologies de développement logiciel. La contribution de cette étude est d'assister les organisations dans leur choix en prenant en compte différents facteurs : la performance, l'adoption et la facilité d'utilisation, les aspects financiers et l'image de la société. A partir de chaque facteur dériveront des facteurs élémentaires qui contribueront à sa réalisation.

Comparée aux travaux précédents, la présente étude offre :

- L'utilisation d'un framework plus riche en facteurs pour faire la comparaison. Ces facteurs adressent des éléments concernant différents aspects du développement logiciel.
- Une comparaison générale de quatre méthodologies comparée à une comparaison spécifique de deux méthodologies telle que dans [2] et [3].
- Une nouvelle approche de sélection basée sur le modèle Balanced Scorecard. Cette approche fournit des recommandations pour le choix d'une méthodologie en équilibrant différents objectifs appartenant aux aspects techniques et aux affaires.
- Le processus de sélection prend en compte les futurs développements et projets de la compagnie et ne se concentre pas seulement sur un seul projet. Par conséquent, le choix des méthodologies pour les futurs projets assure la continuité dans la réalisation des objectifs de la compagnie.

I.3. Structure de cet article

La deuxième section présente et définit les concepts principaux liés aux processus de développement logiciel et à leur comparaison. La troisième section présente les quatre méthodologies à comparer : Rational Unified Process, Cleanroom Software Engineering, Extreme Programming et Open Source Development. Notre contribution se situe dans les deux sections qui suivront. Dans la section quatre, les facteurs de la comparaison sont identifiés sur la lumière de la description des quatre processus fournie dans la section trois. Les facteurs identifiés composant un framework sont employés pour comparer les quatre méthodologies. Les résultats de la comparaison sont donnés ensuite sous forme tabulaire suivis de leur interprétation. La cinquième section présente une nouvelle approche pour la sélection de méthodologies de développement logiciel en utilisant les résultats de la quatrième section. Le modèle proposé est basé sur le Balanced Scorecard, un modèle d'excellence. Ceci est suivi par un ensemble de recommandations concernant l'utilisation de l'approche proposée. Pour illustrer l'applicabilité de cette approche, une étude de cas a été conduite. Celle-ci inclut une description d'une organisation choisie et d'un produit à développer, suivie de recommandations concernant la méthodologie à utiliser.

II. Processus et méthodologies de développement logiciel

II.1. Introduction

Un processus de développement logiciel doit considérer les deux aspects du développement logiciel : l'aspect technique et l'aspect organisationnel. Un processus de développement logiciel est vu en trois dimensions :

- **Méthodologie :** elle peut être regardée comme la manifestation d'introduire la rigueur dans le développement logiciel, au moins en capturant et en normalisant les bonnes pratiques identifiées aussi bien en cherchant une théorie qui soutient le développement.
- **Les gens et l'influence de l'organisation :** qui sont directement reliés à la gestion des activités humaines qui mènent au développement des systèmes logiciels.
- **Technologie :** ou les outils de support.

II.2. Méthodologie

Une méthodologie est un ensemble de directives ou de principes qui peuvent être conçus et appliqués pour couvrir le cycle de vie d'un logiciel sur les plans technique et management [4]. Une description exhaustive d'une méthodologie est donnée dans [5]. Une telle méthodologie devrait inclure les éléments suivants :

- **Rôles :** les gens qui font le travail.
- **Qualifications :** les qualifications requises pour les rôles.
- **Équipes :** les rôles qui travaillent ensemble.
- **Techniques :** les procédures systématiques que les gens emploient pour accomplir leurs activités tout au long d'un cycle de vie entier. Les techniques fournissent également l'aide pour accomplir les livrables.
- **Activités :** les rôles et les équipes deviennent impliqués dans différentes activités, telles que la planification, le codage, la conception et le test.
- **Étapes importantes :** événements marquant le progrès ou l'accomplissement du travail.

- **Valeurs d’équipe :** les valeurs qui conduisent toutes les activités des équipes.
- **Normes :** les conventions que les équipes adoptent.
- **Qualité :** la qualité s'attache aux activités ou aux produits de travail.
- **Livrables :** qu’est-ce qui est construit par les membres des équipes.

II.3. Pourquoi les méthodologies ?

Les chercheurs et les praticiens se sont rendus compte que développer un logiciel n’est pas simplement une question de création de langages et d'outils de programmation efficaces. C'est un effort collectif, complexe et créateur [6]. Un produit logiciel est le résultat d'un système complexe de matériels, de personnes, de logiciels et de procédures. La qualité d'un produit logiciel dépend fortement de ces éléments. Une des manières pour atteindre un résultat plus efficace serait l'adoption des processus de développement comme une base pour le développement de produits logiciels.

Un processus peut inclure un ensemble d'activités exécutées dans une organisation. Il aide à identifier les différentes dimensions du logiciel et les problèmes qui doivent être adressés afin d'établir des pratiques efficaces. Entre autres, le processus de développement logiciel offre les avantages suivants [5] :

- Il permet de faire intégrer rapidement les nouveaux membres dans le processus de développement.
- Il trace les responsabilités. Il indique ce qui ne fait pas partie du travail d'une personne assurant de ce fait la responsabilité dans toutes les étapes.
- Employer une méthodologie cohérente assure les niveaux les plus élevés d'excellence.
- Il permet d’identifier la maturité d’un projet.
- Il est employé pour impressionner les clients. Un processus bien normalisé est un facteur important dans l’attribution de contrats.
- Il évalue l'efficacité des pratiques de développement logiciel dans une organisation, suggère et définit une stratégie d'amélioration.

II.4. Pourquoi les méthodologies diffèrent-elles ?

"One-size-fits-all-methodology" est une expression employée pour décrire une méthodologie qu'une compagnie choisirait pour développer tous ses logiciels; une méthodologie qui est censée résoudre tous ses problèmes logiciels. Après des décennies de recherche dans le domaine du génie logiciel, il est devenu clair qu'il n'aura aucune telle méthodologie. Au lieu de cela, il y avait une variété de méthodologies, chacune est plus appropriée pour un certain projet plus que d'autres.

Dans [7] Robert C. Glass mentionne quatre facteurs ou *"dimensions"* liés au projet selon lesquels une méthodologie peut être accordée au type de projet pour lequel elle est plus adaptée. Ces facteurs sont : la taille, le domaine, la criticité et l'innovation.

Au niveau de l'organisation, les éléments tels que la culture et la dynamique de l'équipe de développement, les préférences du chef de projet, le nombre, les connaissances et l'expérience des membres de l'équipe et les ressources matérielles disponibles peuvent décider sur la méthodologie à employer [8].

II.5. Pourquoi comparer les méthodologies ?

Selon [4], au moins 20 méthodologies différentes concurrencent pour être la meilleure méthode et la liste de méthodologies continue à croître. La liste inclut : Rational Unified Process, Extreme Programming, Crystal, SCRUM, DSDM, Cleanroom et beaucoup d'autres méthodologies. Cette diversité a motivé les chercheurs de comparer les méthodologies existantes afin d'identifier les aspects contradictoires et les différences entre elles. Ceci peut servir pour :

- **Les concepteurs de méthodologies :** la comparaison permet aux *"méthodologistes"* d'apercevoir les aspects manquants de leurs méthodologies et de focaliser leurs efforts pour couvrir ces faiblesses.
- **Les utilisateurs de méthodologies :** les compagnies voulant adopter une méthodologie doivent évaluer et choisir celle qui est plus appropriée pour réaliser leurs objectifs. La comparaison de méthodologies fournit un raccourci pour faire ce choix.
- **L'enseignement :** la comparaison permet aux étudiants du génie logiciel de savoir quelles méthodologies sont disponibles et comment elles se relient entre elles.

III. Les méthodologies de développement Logiciel

III.1. Introduction

Les méthodologies étudiées dans cet article sont Rational Unified Process, Extreme Programming, Cleanroom Software Engineering et Open Source Development. RUP a été choisie parce qu'elle est employée dans le monde entier par des milliers de compagnies. XP est une méthode révolutionnaire de développement logiciel. Elle est considérée comme rivale de RUP. Elle essaye de résoudre certains des inconvénients de RUP comme il est mentionné par ses auteurs. Open Source Development est une tendance internationale qui est totalement différente des approches *"classiques"* de développement logiciel. En outre, un nombre croissant de compagnies la soutient. Cleanroom Software Engineering est une approche qui a pris ses principes de l'industrie des semi-conducteurs et promet la production de logiciels de haute qualité. Ces quatre processus de développement logiciel représentent le courant principal dans le développement logiciel.

III.2. Rational Unified Process (RUP)

RUP est un processus de développement logiciel. Il fournit une approche disciplinée au développement logiciel en assignant les tâches et les responsabilités dans une organisation de développement logiciel. Son but est d'assurer la production d'un logiciel de haute qualité qui satisfait les besoins de ses utilisateurs dans un délai et un budget prévisibles [9].

RUP est un produit qui est développé et maintenu par Rational Software. Il est intégré avec sa suite d'outils de développement logiciel. Il est fourni par IBM sur CD-ROM ou via Internet [9]. RUP est employé par une grande variété de compagnies dans différents secteurs industriels.

Un ensemble de principes fondamentaux qui soutiennent le développement itératif représente l'*"esprit de RUP"*. Ces principes ont été collectés à partir d'un énorme nombre de projets réussis. Ces principes sont [10] :

- Attaquer les risques principaux aussi tôt et sans interruption.
- S'assurer de fournir de la valeur à votre client.
- Rester concentré sur l'exécutable.
- S'adapter aux changements tôt dans le projet
- Se baser sur une architecture exécutable dès le début.

- Construire votre système avec des composants.
- Travailler ensemble en tant qu'une seule équipe.
- Faire de la qualité une façon de vivre.

III.3. Extreme Programming (XP)

Extreme Programming (XP) est l'une des méthodes *"agiles"* les plus connues. XP a été principalement développée par Kent Beck et Ward Cunningham. Le cycle de vie de XP est composé de cinq phases [11] : l'exploration, la planification, l'itération, la mise en production, la maintenance et la mort du projet.

XP est vue comme un ensemble de pratiques destinées à organiser le travail d'une équipe de développement. Elle est basée sur quatre valeurs fondamentales [12] : la communication, la simplicité, le retour d'information (feedback) et le courage. Autour de ces valeurs sont organisées les douze pratiques suivantes : livraisons fréquentes, planification itérative, client sur site, rythme durable, conception simple, remaniement (refactoring), tests unitaires, programmation en binômes, responsabilité collective du code, règles de codage, métaphore et intégration continue.

III.4. Cleanroom Software Engineering

Cleanroom est une méthode basée sur la théorie pour le développement et la certification des systèmes logiciels de haute fiabilité sous le contrôle de qualité statistique. L'objectif principal de Cleanroom est le développement d'applications qui montrent zéro défaut en marche. Cleanroom insiste sur la discipline et la rigueur et privilégie la prévention des erreurs plutôt que leur correction [13].

Cleanroom combine des méthodes formelles de spécification de logiciels, de conception, et de vérification d'exactitude avec le test d'utilisation statistique pour la certification de la qualité [14].

Les principes de base de Cleanroom sont les suivants [15]:

- Développement incrémental sous le contrôle de qualité statistique.
- Développement de logiciel basé sur des principes mathématiques.
- Les tests logiciels basés sur des principes statistiques.

Dans un processus de développement Cleanroom, trois équipes participent à l'exécution des activités [16] :

- L'équipe de spécification crée les spécifications de haut niveau du système et du logiciel.
- Les équipes de développement modélisent et implémentent les incréments, mais n'exécutent pas le code résultant.
- L'équipe de certification utilise le test d'usage statistique dans l'environnement du système pour conduire la qualité du logiciel aux zéro défaut avec une probabilité élevée.

III.5. Open Source Development

Le concept du développement Open Source a récemment gagné de la popularité comme une approche radicale au développement logiciel. L'objectif de n'importe quel projet Open Source est le partage du code source d'un produit logiciel en le rendant non-propriétaire et de le partager avec n'importe quel utilisateur en utilisant une licence Open Source.

Un projet Open Source est lancé par un ou plusieurs développeurs. Ils codent et fixent le code source jusqu'à ce qu'il semble acceptable et alors ils le passent à la communauté d'utilisateurs.

Une itération typique d'un projet Open Source suit les étapes suivantes [4]:

- Le développeur fait la conception et la génération de code, à titre individuel ou bien dans une équipe.
- Les développeurs ou les utilisateurs font des déboguages et des tests concurrents.
- Les nouvelles fonctionnalités et les bogues identifiés sont passés au propriétaire du projet.
- Une version est mise en place en se basant sur les bogues identifiés et toutes les nouvelles fonctionnalités demandées.
- Quand une nouvelle version est produite, une liste de tâches est distribuée à la communauté d'utilisateurs, cherchant des membres pour réaliser volontairement les tâches sur la liste.

III.6. Conclusion

Les quatre méthodologies étudiées présentent des différences et partages des soucis communs. Ces quatre méthodologies peuvent être classifiées en utilisant trois aspects principaux qui influencent le comportement de chaque méthodologie. Ces trois facteurs sont les suivants :

L'architecture

- La première dichotomie contraste un *"framework"* avec une *"méthodologie de développement"*. Le framework se rapporte aux divers segments du projet et la méthodologie de développement est le moyen de passer d'un segment à un autre. RUP est un framework tandis que Cleanroom, Open Source et XP sont considérées comme des méthodologies de développement. Une telle classification a un effet sur l'adoption et la facilité d'utilisation de la méthodologie.

La modélisation

- Différentes techniques de modélisation sont employées pour documenter le processus de développement. Ceci varie de la spécification formelle (Cleanroom) à la modélisation objet (RUP) et la modélisation agile (XP). L'utilisation de telles techniques exige d'autres ressources telles que les outils de support et la formation pour leur utilisation. D'autres facteurs tels que le retour sur investissement et le budget alloué devraient être pris en compte.

Le processus

- Une autre dichotomie est fournie par deux grandes familles de processus: les processus lourds ou les processus guidés par les plans contre les processus légers ou les processus agiles. Les méthodologies lourdes tentent de planifier une grande partie du projet sur une longue durée (RUP et Cleanroom) tandis que les processus légers ou les processus agiles (XP et Open Source) sont conçus pour s'adapter aux changements des besoins du client. Les facteurs tels que la discipline, la communication et le dynamisme dans l'équipe de développement sont définis par les principes fondamentaux de la méthodologie.

IV. La comparaison de méthodologies

IV.1. Introduction

La comparaison courante vise à fournir un framework de facteurs qui guide les compagnies dans leur choix de méthodologies. Dans cette étude, nous avons limité notre comparaison à quatre méthodologies qui représentent le courant principal dans le développement logiciel : Rational Unified Process, Cleanroom Software Engineering, Extreme Programming et Open Source

Development. Des facteurs primaires ont été choisis dans la présente étude par rapport au projet, à la méthodologie et à l'organisation :

- **Facteurs du projet :** la qualité demandée, l'innovation, le domaine d'application et la criticité;
- **Facteurs de la méthodologie :** les outils de support, le retour sur investissement de l'amélioration du processus de développement logiciel, l'amélioration du processus de développement logiciel et l'adoption et la personnalisation;
- **Facteurs de l'organisation :** la taille, la discipline et les ressource disponibles.

Le choix des facteurs a été basé sur les informations fournies dans les sections II et III. Ces facteurs représentent les éléments les plus importants qui peuvent affecter le choix de la méthodologie. Chaque facteur a été évalué indépendamment des autres et contrasté dans le contexte des quatre méthodologies. Les facteurs choisis et la comparaison sont présentés dans ce qui suit :

IV.2. Projet

IV.2.1. Qualité du logiciel

1. Rational Unified Process

La vérification de la qualité sans interruption est l'une des six meilleures pratiques adoptées par RUP. La qualité devrait être passée en revue en ce qui concerne différentes dimensions : fiabilité, fonctionnalité, performance et rentabilité; et différents types de tests : test unitaire, test d'intégration, test du système et test d'acceptation. RUP aide l'utilisateur dans la planification et l'exécution de tous ces types de tests en fournissant les techniques et les outils appropriés.

2. Extreme Programming

XP insiste sur la production d'un logiciel de haute qualité. Elle assure la qualité en se basant sur ses deux aspects : les besoins du client et le taux de défaillances. La qualité est améliorée dans tout le processus de développement en fixant les erreurs si tôt et en fournissant un feedback constant sur le produit.

3. Cleanroom Software Engineering

Cleanroom est axée sur la prévention des erreurs plutôt que sur leur correction. En utilisant un compromis de spécification formelle, de test statistique et de la vérification, nous pouvons atteindre un logiciel qui s'approche du zéro erreur.

4. Open Source Development

Il n'y a aucun test système. La qualité est réalisée par le test continu et le versionnement fréquent. On rapporte que les logiciels Open Source ont moins de bogues que les produits développés d'une manière traditionnelle. Cependant, les produits Open Source souffrent de la mauvaise expression des besoins du client. En fait, il n'y a aucun vrai besoin à part les fonctionnalités de base. Les besoins sont sujets de changement pendant le processus de développement. Les fonctionnalités du produit final peuvent être totalement différentes de ce qui a été indiqué.

IV.2.2. Innovation

1. Rational Unified Process

La première étape dans le développement d'un nouveau produit avec RUP est d'étudier la faisabilité du projet. Cette évaluation aura comme résultat un plan d'implémentation qui nous indique toutes les ressources nécessaires pour accomplir le projet. Concernant les anciens systèmes, RUP n'adresse pas explicitement comment les traiter. Une extension de RUP a été fournie par Ronin International, inc. [17]. L'extension proposée supporte les activités de maintenance et d'après-développement.

2. Extreme Programming

Le principe du *"client sur site"* permet aux équipes XP d'avoir une idée claire sur les fonctionnalités du système. Le client agit en tant que membre de l'équipe de développement et fournit les détails demandés. La pratique Refactoring démontre la force de XP en manipulant les systèmes existants. Avec des développeurs expérimentés, le Refactoring est un excellent outil pour améliorer le code source de ces systèmes.

3. Cleanroom Software Engineering

La manipulation du code existant et la réécriture des systèmes existants sont discutées dans [18]. La réécriture est limitée aux systèmes développés en utilisant l'approche Cleanroom. Ceci inclut les petites modifications, les réécritures partielles et l'ajout de nouveaux composants. L'auteur fournit dans la même référence un certain nombre de pratiques et de techniques pour améliorer la qualité des systèmes existants.

4. Open Source Development

La plupart des logiciels Open Source sont des applications systèmes (systèmes d'exploitation, navigateurs Web, compilateurs, etc.) où les développeurs ont suffisamment d'informations sur les besoins et l'architecture du produit [19]. Avec les nouveaux produits et des communautés distribuées, il serait extrêmement difficile d'exprimer les besoins pour être compris par les programmeurs. La modification du code source est au contraire l'une des grands avantages du développement Open Source par rapport aux autres approches de développement.

IV.2.3. Domaine du projet

1. Rational Unified Process

RUP peut être employée dans divers domaines d'application et pour les grands et les petits projets. Elle est employée par de nombreuses compagnies dans différents domaines: la télécommunication (Ericsson et Alcatel), le transport, l'espace et la défense (British Aerospace), la fabrication (Xerox) et la finance.

2. Extreme Programming

Bien réussie dans beaucoup de domaines d'application, aucune limite claire n'a été encore identifiée à l'applicabilité de XP.

3. Cleanroom Software Engineering

Cleanroom est la méthode la plus appropriée aux applications critiques. Le terme *"critique"* se rapporte aux applications où les erreurs peuvent causer la perte de vie ou une grande perte financière. L'espace et la défense, la télécommunication, et les applications systèmes sont les domaines principaux où Cleanroom a été appliquée avec succès.

Cleanroom est employée au niveau des compagnies commerciales mais sur une échelle très limitée.

4. Open Source Development

La nature du développement Open Source lui fait une cible des grands projets avec des communautés distribuées. La plupart des projets Open Source sont des grands outils de développement et des produits Web. Open Source peut s'adapter pour des domaines spécifiques mais la principale difficulté est de trouver et de motiver des développeurs habiles.

IV.2.4. Criticité du projet

1. Rational Unified Process

Les buts primaires des méthodes guidées par les plans telles que RUP sont la prévisibilité, la stabilité et l'assurance élevée. Avec son approche basée sur le risque, ses plans et spécifications complètement documentées, RUP convient mieux pour le développement des systèmes exigeant une haute assurance et une haute sûreté de fonctionnement. Avec les produits moins critiques, RUP devient lente et montre moins d'efficacité due à la documentation excessive qui devrait être produite.

2. Extreme Programming

XP est non encore testée avec les produits critiques. Les difficultés principales remontent de la conception simple et du manque de la documentation [12]. Cependant, avec les systèmes non critiques, XP semble donner de meilleurs résultats que les méthodes guidées par les plans.

3. Cleanroom Software Engineering

Cleanroom a été conçue pour développer des applications qui ne montrent aucun échec en service. Le développement par incrément sous un contrôle statistique du processus de développement avec la spécification et la vérification formelles du logiciel maintiennent le contrôle sur le projet. Ceci réduit le risque et donne plus de productivité et plus de fiabilité dans le développement logiciel.

4. Open Source Development

Les projets Open Source sont entrepris par des volontaires sans plan de projet, sans programme de développement, sans conception architecturale ou détaillée et sans liste de livrables [20]. Les limitations géographiques entre les développeurs ajoutent plus d'incertitudes au processus de développement et le rendent moins prévisible avec moins d'assurance.

IV.3. Méthodologie

IV.3.1. Outils de support

1. Rational Unified Process

RUP fournit un ensemble d'outils qui aident à automatiser les activités du processus de développement. Ces outils sont groupés dans une suite qui s'appelle *"IBM Solutions"*. Elle inclut RUP Builder pour composer et éditer votre propre configuration de RUP, Rational XDE pour la modélisation visuelle et Rational ClearCase pour la gestion de la configuration.

2. Extreme Programming

XP ne fournit et ne recommande aucun outil de support. Il appartient à l'utilisateur de choisir les outils appropriés.

3. Cleanroom Software Engineering

Un ensemble d'outils appelé toolSET est utilisé pour supporter les activités de spécification, de développement et de certification. toolSET inclut les outils suivants:

- **Certification Assistant:** il produit automatiquement des cas de test à partir des distributions de probabilité d'utilisation et effectue l'analyse statistique des résultats de tests.
- **toolSET_Certify :** un outil CASE pour le test d'utilisation statistique.
- **CleanTest :** il produit des cas de test statistiques basés sur le profil d'entrée.

4. Open Source Development

Différents outils libres sont utilisés comme les navigateurs Web (Netscape), les éditeurs de texte (Emacs), les compilateurs (GNU Make) et les outils de gestion de la configuration (CVS), etc.

IV.3.2. Retour sur investissement de l'amélioration du processus de développement logiciel

1. Rational Unified Process

Les meilleures pratiques du développement logiciel recommandées par RUP sont des pratiques prouvées qui supportent les décisions techniques et commerciales. D'autre part, l'utilisation des outils intégrés au produit RUP (IBM Solutions) peut fournir un retour sur investissement élevé en automatisant les tâches.

2. Extreme Programming

XP n'effectue pas l'analyse financière du retour sur investissement pour déterminer la répartition des ressources. Les équipes de XP *"font la chose la plus simple qui fonctionne"*. Elles essayent d'éviter les coûts de fonctionnalités non encore nécessaires. Par la simplicité et la livraison rapide, XP réduit les délais de développement ce qui mène à réduire les dépenses de fonctionnement.

3. Cleanroom Software Engineering

Cleanroom réalise un meilleur retour sur investissement en évitant la correction des erreurs d'après-production (trouver et corriger les erreurs, suivre les problèmes et distribuer les correctifs). Ceci réduit le double travail et résulte en une réduction remarquable dans les coûts directs de la correction d'erreurs durant toute la durée de vie du produit.

4. Open Source Development

Les coûts de formation pour migrer vers Open Source réduisent le retour sur investissement pour une seule application. Si nous prolongeons l'évaluation sur toute la durée de vie de l'application, ou plus, après avoir développé d'autres applications, le coût de formation devient marginal et nous pouvons tirer avantage du travail gratuitement fait par d'autres programmeurs.

IV.3.3. Possibilités de l'amélioration du processus de développement logiciel

1. Rational Unified Process

Les compagnies utilisant RUP peuvent atteindre les niveaux 2 et 3 de CMM en complétant quelques aspects de gestion du projet [21]. Peu de travail a été fait pour évaluer RUP par rapport aux niveaux 4 et 5 de CMM. La plupart des soucis du CMM à ces deux niveaux sont liés à l'organisation. Des nouvelles procédures devraient être mises en application pour satisfaire les secteurs clés correspondants.

2. Extreme Programming

XP satisfait la plupart des secteurs clés des niveaux 2 et 3 de CMM. Les secteurs clés absents (la gestion des contrats pour le niveau 2 et le programme de formation et la gestion intégrée de logiciel pour le niveau 3) devraient être adressés en utilisant un support de gestion approprié. Au-delà du niveau 3, XP ignore ou couvre partiellement les secteurs clés de CMM. Les organisations voulant atteindre les niveaux 4 et 5 du CMM devraient utiliser leurs propres ressources pour satisfaire ces secteurs clés.

3. Cleanroom Software Engineering

Cleanroom et CMM ont été développés par Software Engineering Institute (SEI) et ils partagent les mêmes soucis. Le SEI a développé un modèle de référence de Cleanroom (CRM) qui fournit un framework pour développer un projet ou un processus de développement logiciel d'une organisation. Une fois adaptée, Cleanroom implémente une majorité de CMM. En outre, Cleanroom adresse des processus qui n'ont pas de secteurs clés correspondants dans le CMM [22].

4. Open Source Development

Un processus formel appelé Open Source Maturity Model (OSMM) [23] est employé pour évaluer le niveau de maturité des logiciels Open Source mais pas du processus du développement. Un certain travail a été effectué pour relier le développement Open Source avec le CMM. Des évaluations ont été faites pour des compagnies ayant un processus défini et voulant migrer vers le développement Open Source. Une étude de cas a été présentée dans [24]. Elle a conclu que la maturité du processus a passé du niveau 3

au niveau 2, alors qu'il est possible d'atteindre de nouveau le niveau 3 une fois l'équipe de développement s'habitue aux rajustements organisationnels.

IV.3.4. Adoption et personnalisation

1. Rational Unified Process

RUP pourrait être adoptée entièrement ou partiellement comme indiqué dans [9]. Elle doit être configurée et adaptée en fonction du contexte et des besoins spécifiques de l'organisation avant son implémentation complète. Le processus d'adoption est un programme itératif de cinq étapes comprenant : l'évaluation, la planification, la configuration et l'adaptation, l'implémentation et l'évaluation.

2. Extreme Programming

Pour adopter XP vous devrez l'utiliser. Ses pratiques devraient être adoptées graduellement. Vous devrez adopter une pratique à la fois, toujours en adressant le problème le plus urgent pour votre équipe. Une fois traité, continuez au prochain problème. Dans la pratique, on peut rarement adopter toutes les pratiques.

3. Cleanroom Software Engineering

Selon [25], Cleanroom peut être adoptée de trois manières : partielle, complète ou avancée. L'auteur suggère une approche d'implémentation par phase en abordant d'abord les principes fondamentaux de Cleanroom. Avec l'accroissement de l'expérience et de la confiance de l'équipe de développement, la précision et la rigueur accrues peuvent être introduites dans une implémentation complète de Cleanroom. Finalement, une implémentation avancée peut être introduite pour optimiser le processus Cleanroom.

4. Open Source Development

Adopter le développement Open Source veut dire ouvrir le code source de votre application sous une certaine licence. Vous devrez d'abord lancer un projet avec une idée qui vous intéresse. Ensuite, vous devrez motiver les participants et produire des versions aussi tôt et aussi souvent que possible. La difficulté dans l'adoption du développement Open Source est de trouver et de motiver les participants et d'assurer la continuité du projet.

IV.4. Organisation

IV.4.1. Taille de l'équipe

1. Rational Unified Process

Deux membres ou plus. Il n'y a aucune limite supérieure pour les équipes RUP. RUP s’adapte mieux aux grands projets; plus l'équipe est grande, plus le processus est efficace.

2. Extreme Programming

XP est visée par les équipes de petites et moyennes tailles. Une équipe XP idéale devrait être comprise entre trois et vingt membres. Les équipes avec une taille supérieure sont possibles mais tendent à échouer à appliquer les pratiques et les principes de XP.

3. Cleanroom Software Engineering

Pour les équipes de petite taille entre six et dix membres. Avec les grands projets, les équipes peuvent être subdivisées en sous-équipes.

4. Open Source Development

Les programmeurs contribuent librement aux projets Open Source. Ceci constitue des équipes distribuées de volontaires. Le nombre des membres peut s'étendre jusqu'à quelques milliers.

IV.4.2. Discipline

1. Rational Unified Process

RUP impose la discipline comme facteur essentiel pour le succès de n'importe quel projet. Chaque membre fait une tâche bien définie avec une connaissance limitée sur ce que les autres font jusqu'à ce que le processus devienne normalisé. Après la normalisation du processus, le changement devient difficile et lent.

2. Extreme Programming

XP convient mieux aux environnements turbulents avec des changements fréquents [26]. Elle donne de meilleurs résultats avec des gens qui veulent collaborer. Les développeurs sentent confortable et autoritaire et ayant un certain degrés de liberté.

3. Cleanroom Software Engineering

Cleanroom considère le développement logiciel comme une discipline de la technologie et pas comme un art ou un métier. Ceci donne peu de liberté de travail aux gens. Le processus de développement est réalisé d'une manière très disciplinée.

4. Open Source Development

Le système est développé par un grand nombre de volontaires. Le travail n'est pas assigné. Au lieu de cela, les gens eux-mêmes choisissent les tâches qui leur intéressent.

IV.4.3. Ressources disponibles

1. Rational Unified Process

RUP est un produit propriétaire. Adopter RUP exige l'investissement de beaucoup de ressources: l'achat de la description du processus de développement et des outils de support et les coûts de formation pour utiliser ces outils. Des coûts additionnels peuvent être exigés pour adapter RUP pour les besoins de la compagnie.

2. Extreme Programming

La formation et la consultation sont les seuls besoins.

3. Cleanroom Software Engineering

Les outils de support devraient être acquis. La formation est exigée pour apprendre les techniques de spécification et de vérification.

4. Open Source Development

Aucun coût n'est imposé. Certains projets Open Source peuvent être financés par des compagnies pour encourager les programmeurs à participer.

IV.5. Résultats et interprétation

Les tableaux suivants récapitulent la comparaison faite dans la section précédente. Les trois tableaux montrent le niveau du souci fourni par chaque méthodologie pour les différents facteurs. L'évaluation donnée varie de très bas à très haut.

Caractéristiques du projet

Projet	Qualité	Innovation	Domaine	Criticité
RUP	Haut	Haut	Très Haut	Très Haut
XP	Haut	Haut	Haut	Moyen
Cleanroom	Très Haut	Bas	Très Bas	Très Haut
OSD	Moyen	Moyen	Moyen	Bas

Caractéristiques de l'organisation

Organisation	Taille de l'équipe	Discipline	Ressources
RUP	Très Haut	Haut	Très Haut
XP	Moyen	Bas	Bas
Cleanroom	Bas	Très Haut	Haut
OSD	Très Haut	Très Bas	Très Bas

Caractéristiques de la méthodologie

Méthodologie	Outils de support	Retour sur investissement	Amélioration du processus de développement logiciel	Adoption et personnalisation
RUP	Haut	Haut	Haut	Haut
XP	Bas	Haut	Haut	Moyen
Cleanroom	Très Haut	Haut	Très Haut	Bas
OSD	Très Haut	Très Haut	Bas	Moyen

L'interprétation des tableaux précédents et quelques recommandations générales sont décrites ici:

- **Qualité :** Cleanroom convient mieux pour les projets exigeant un niveau de qualité très élevé. RUP et XP peuvent également assurer un niveau élevé de

qualité. Open Source peut fournir un code avec un taux bas de bogues mais souffre de la pauvre expression des besoins.

- **Innovation :** RUP et XP traitent mieux les anciens systèmes. Open Source est bonne pour modifier les anciens systèmes mais l'absence de la documentation limite cet avantage. Avec Cleanroom on ne peut modifier que les systèmes développés en utilisant les mêmes techniques de spécification et de certification.
- **Domaine d'application :** RUP et XP montre une diversité d'application. Open Source a été employée dans une gamme limitée des produits (Web et systèmes d'exploitation) et doit être testée sur d'autres domaines. Cleanroom vise un domaine restreint d'applications tel que les systèmes en temps réel.
- **Criticité :** Plus le système est critique, plus de densité est nécessaire et une méthodologie plus lourde est exigée (RUP et Cleanroom) avec un plan de développement détaillé. XP et Open Source conviennent mieux pour les applications moins critiques et donnent de meilleurs résultats comparées aux méthodologies guidées par les plans.
- **Communication et discipline :** XP et Open Source sont fondées sur des facteurs humains. La motivation et la communication libre entre les équipes de développements sont importantes pour le succès du projet. RUP et XP sont fondées plutôt sur la discipline et la bonne planification.
- **Taille de l'équipe :** RUP et Open Source s'adaptent mieux avec les grandes équipes. XP et Cleanroom visent habituellement les équipes de petites tailles.
- **Ressources disponibles :** RUP et Cleanroom exigent un grand investissement particulièrement pour la formation et les outils de support. XP et Open Source au contraire n'exigent pas beaucoup de ressources.
- **Outils de support :** RUP, Cleanroom et Open Source fournissent les outils de support nécessaires pour assister les activités de développement. XP se base surtout sur l'habileté des développeurs et ne fournit pas un tel support.

- **Retour sur investissement :** Open Source fournit un retour sur investissement élevé sur le long terme. RUP, XP et Cleanroom fournissent plus ou moins un retour sur investissement élevé.
- **Amélioration du processus de développement logiciel :** L'application directe de Cleanroom mène à un niveau très élevé de maturité. RUP et XP peuvent atteindre des niveaux de maturité élevés avec des arrangements spéciaux. Open Source ne s'intéresse pas à la maturité du processus de développement et ne permet pas donc d'atteindre des niveaux élevés.
- **Adoption :** RUP montre plus de flexibilité dans son adoption. Il fournit des recommandations pour son adaptation aux besoins de la compagnie. Cleanroom fournit suffisamment de documentation sur son adoption mais dans la pratique il est très difficile d'appliquer ses techniques et ses pratiques. Open Source et XP n'exigent pas d'arrangements spéciaux. Leur adoption est une question de bonne volonté et d'acceptation au sein de la compagnie.
- **Besoins :** RUP et Cleanroom sont conçues pour les projets dont les besoins sont stables et XP et OSD pour les projets avec des besoins qui changent rapidement.
- Il est possible d'adopter une version légère de RUP qui inclut un sous-ensemble de processus de RUP. Beaucoup de travail a été fait pour montrer comment ajuster RUP pour les petites équipes. Un certain travail a été effectué dans le sens inverse: comment utiliser XP pour les grands projets mais reste à tester dans la pratique.
- Avec une grande équipe, différentes sous-équipes peuvent être arrangées pour utiliser différentes méthodologies. Chaque sous-équipe emploie la méthodologie qui convient mieux à sa partie du projet.
- Pour un grand projet, XP peut être employée par exemple dans les équipes tandis que RUP entre les équipes. XP assure la bonne communication dans les équipes et RUP fournit la planification appropriée et assure la discipline.
- Cleanroom peut être utilisée pour développer les parties critiques d'un projet et RUP ou XP ou OSD pour le reste.

V. Sélection de méthodologies

V.1. Introduction

Dans cette partie, nous présentons une nouvelle approche pour la sélection de méthodologies de développement logiciel. L'approche présentée est basée sur le modèle Balanced Scorecard (BSC). Elle combine l'aspect technique et l'aspect commercial du développement logiciel.

Le modèle Balanced Scorecard a été utilisé comme un conducteur pour la sélection. La description du BSC et la manière de l'utiliser pour choisir une méthodologie pour un projet sont présentées dans la section suivante.

V.2. L'approche de sélection

L'approche proposée pour la sélection est guidée par le Balanced Scorecard: un modèle d'aide à la décision bien connu. Le Balanced Scorecard est présenté dans la section V.2.1. Dans la section V.2.2 nous présentons le procédé de sélection.

V.2.1. Le Balanced Scorecard

Le Balanced Scorecard (BSC) est une approche récente pour mesurer la performance d'une entreprise et d'aide à la décision stratégique. Cette approche était fondée au sein de la communauté industrielle et a gagné un intérêt considérable. Le BSC groupe des indicateurs de même type en ensembles appelés perspectives. Le modèle BSC de Kaplan et de Norton [46] définit quatre perspectives complémentaires qui peuvent mesurer la performance d'une compagnie :

- **Perspective Finance :** Comment sommes nous perçus par nos actionnaires?
- **Perspective Client :** Comment sommes nous perçus par nos clients?
- **Perspective Processus Internes :** Quels sont les processus internes dans lesquels nous devons exceller pour réussir?
- **Perspective Apprentissage Organisationnel :** Comment optimiser notre capacité à changer et à nous améliorer?

Ces quatre perspectives équilibrent les objectifs à court et à long terme. Elles concentrent sur tous les deux : les résultats financiers et les possibilités concurrentielles à long terme.

V.2.2. Le Balanced Scorecard comme conducteur de la sélection

Au-delà de son utilisation normale comme outil pour la gestion stratégique, le BSC peut être employé pour comparer les méthodologies de développement logiciel. Avec ses indicateurs de performance multicritères, nous pouvons évaluer les méthodologies de développement logiciel face aux quatre perspectives définies par le BSC. Pour un projet donné, la compagnie peut avantager une ou plusieurs perspectives par rapport aux autres. En classifiant les différents facteurs affectant le développement logiciel par rapport aux quatre perspectives du BSC, nous obtenons une image claire sur les éléments à prendre en compte pour atteindre les objectifs définis par la compagnie. La figure 1 présente une telle classification. Cette figure se rapporte à la valeur que nous pourrions tirer d'une méthodologie de développement logiciel. Les caractéristiques du projet, de l'organisation et de la méthodologie ont été réparties sur les quatre perspectives du BSD en montrant en même temps les points communs entre les quatre perspectives. Les différentes caractéristiques sont groupées comme suit :

- **Perspective Finance :** inclut le coût pour les outils de support et de la formation, le retour sur investissement et les ressources disponibles.
- **Perspective Client :** c'est le moyen de rester compétitif en satisfaisant les clients et en améliorant l'image de la compagnie. La qualité et la maturité du processus de développement logiciel sont les facteurs principaux qui l'affectent.
- **Perspective Processus Internes :** inclut l'adoption, la taille de l'équipe de développement, la facilité d'utilisation, les outils de support, l'expérience du personnel, l'innovation et le domaine d'application.
- **Perspective Apprentissage Organisationnel :** elle est reflétée dans les facteurs suivants: la qualité, la communication et le dynamisme, et le retour sur investissement de l'amélioration du processus de développement logiciel.

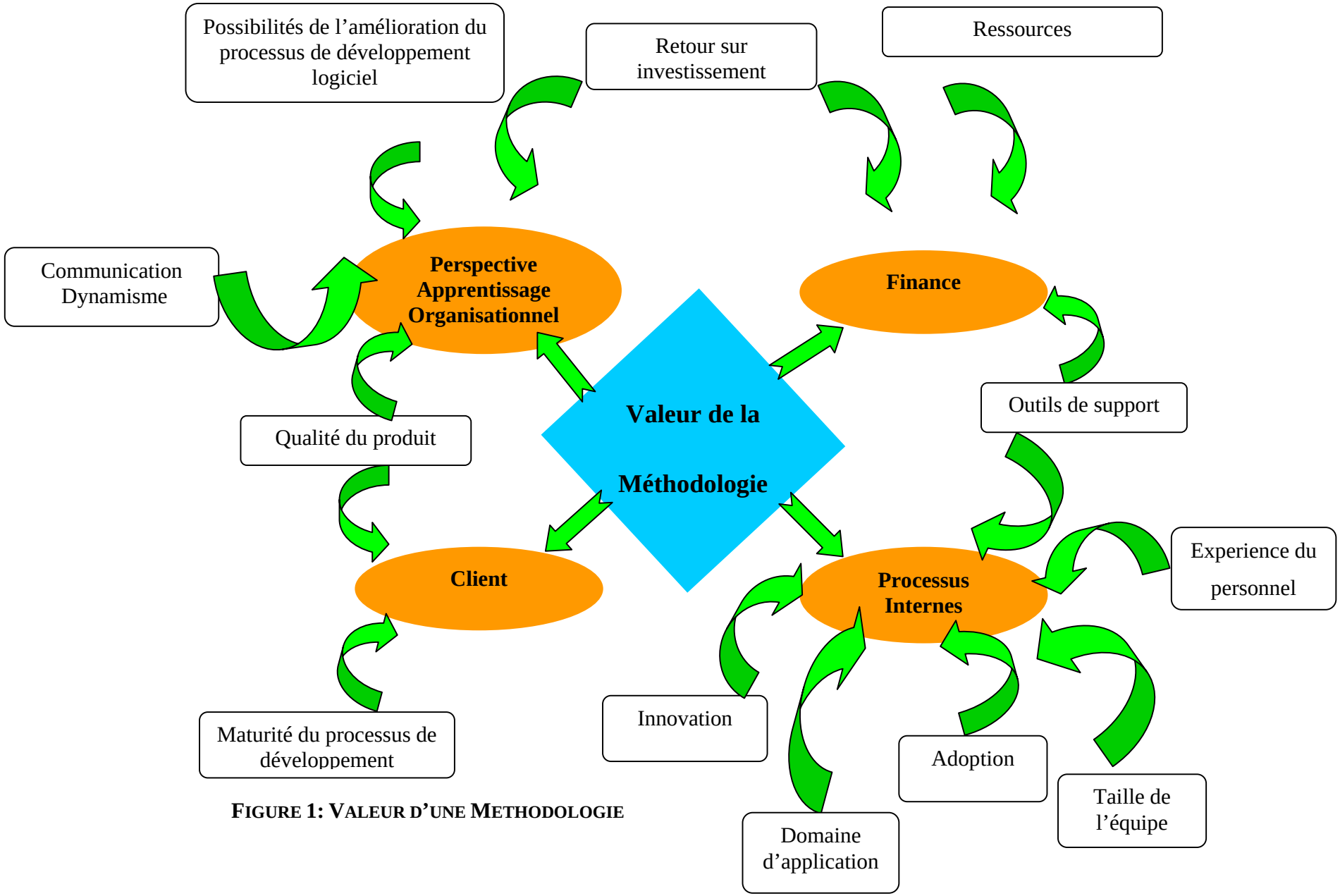

FIGURE 1: VALEUR D'UNE METHODOLOGIE

V.2.3. Recommandations Générales

Le choix d'une méthodologie exige recueillir assez d'informations sur le projet, l'organisation et les méthodologies disponibles et leurs portées d'utilisation. Nous pouvons alors faire correspondre notre projet avec une méthodologie appropriée. Les informations liées à l'organisation devraient être étudiées d'abord: la possibilité d'améliorer le processus de développement logiciel, les outils de support pour la gestion de projet, les qualifications, les connaissances et la bonne volonté des membres de l'équipe de développement, la discipline ou la culture et les préférences du chef de projet, etc. La prochaine étape devrait adresser les éléments du projet: la qualité exigée, l'innovation, le domaine d'application, la criticité du projet et les ressources demandées, etc. A ce point là, à l'aide du framework fourni, le chef du projet peut faire son choix selon les objectifs et les préférences de l'organisation et du projet. Il adoptera la méthodologie qui contribue le mieux à l'accomplissement des perspectives qui correspondent aux objectifs fixés et puis il l'adaptera aux besoins de la compagnie.

V.3. Étude de cas

Avec les considérations données dans la section IV et les recommandations fournies dans la présente section, nous avons conduit une étude de cas dans une compagnie de développement logiciel pour démontrer l'applicabilité du framework fourni. Nous décrirons d'abord l'organisation: l'équipe de développement et les processus existants dans l'organisation. Nous décrirons ensuite le produit à développer: ses fonctionnalités, la qualité requise et sa criticité à tous les deux: la compagnie et le client. Par rapport aux différents facteurs analysés dans les sections précédentes, nous recommandons une méthodologie parmi les quatre méthodologies étudiées pour ce projet. Des arguments seront fournis pour justifier notre choix.

V.3.1. L'organisation

Macro Data World ou *"MDWorld"* est une compagnie de développement logiciel spécialisée dans les systèmes d'aide à la décision (SAD). MDWorld a été créée en 2000 par un groupe d'ingénieurs d'informatique en collaboration avec MCS Company (Mediterranean Consult and Service Company), le leader de la consultation en Algérie. MDWorld a été créée pour compléter et étendre les activités de MCS Company. Son but était de fournir des solutions efficaces pour ses clients telles que le développement de logiciels et l'étude et l'évaluation des systèmes d'information existants.

MDWorld est gérée par un personnel composé d'un directeur général, un directeur technique, un directeur des ventes et un directeur de ressources. MDWorld est structurée en trois départements principaux :

- **MDWCS** est responsable du suivi de projets.
- **MDWRD** suit les nouvelles techniques et technologies afin de rester à jour avec le progrès dans leur secteur de développement.
- **MDWIS** étudie et développe les systèmes logiciels pour les clients.

MDW avait développé du logiciel depuis 2000 où l'équipe de développement s'est composée de 4 développeurs. Depuis lors, l'équipe de développement a grandi et est composée actuellement de 7 personnes, avec 5 qui sont directement impliquées dans le développement logiciels. L'équipe de développement est composée d'un groupe d'ingénieurs expérimentés travaillant ensemble depuis 2000. Ceci a développé un bon sens de communication chez le groupe et a leur permis de partager les connaissances d'une manière efficace. Les cinq ans d'expérience ont permis à la compagnie d'établir quelques processus de développement et de gestion mais il n'y avait aucune description formelle des activités à faire ni d'historique des projets précédents. Les activités de développement sont tenues dans un appartement de quatre chambres qui est également employée comme siège pour l'administration de la compagnie.

MDW a été le développeur de plusieurs projets depuis 2000. Les organismes suivants sont certains des clients de MDW:

- Ministère Algérienne de la Commerce.
- Organisation des Nations Unies pour l'Alimentation et l'Agriculture.
- Biwater France SA
- VitaJus
- Ministère Algérienne de l'Agriculture et du Développement Rural.

V.3.2. Le projet

La mission de l'équipe de développement était de créer un outil d'analyse intelligent basé sur les techniques du Datawarehousing pour aider les compagnies à stocker leurs informations et à

fournir des analyses détaillées des données recueillies afin de soutenir les processus de décision. Le produit sera appelé Maestro. Il doit être développé comme un produit COTS. Il doit être configuré avant d'être employé par les compagnies.

La plate-forme Dot Net a été choisie pour développer ce produit. L'équipe de développement avait l'habitude de développer ses produits sous l'environnement Delphi et a décidé maintenant de migrer vers la technologie récente de Microsoft. Les membres de l'équipe de développement ont peu d'expérience avec la technologie Dot Net. Une formation spéciale est exigée pour s'adapter avec le nouvel environnement. La compagnie a programmé une session de formation pour tous les membres de l'équipe et a alloué un budget spécial pour cette formation.

Fonctionnalités du produit :

Le produit est censé assurer les fonctions suivantes :

- Consolider et centraliser toutes les données disponibles.
- Fournir un tableau de bord sur les activités de l'entreprise.
- Employer différents modèles pour l'analyse de données, et
- Assurer la diffusion de l'information au niveau de la compagnie.

Il y a deux ans, la compagnie a lancé un processus d'évaluation du processus de développement interne. Le processus a été entrepris par des étudiants du département d'Informatique de l'université de BOUMERDES en collaboration avec quelques membres de la compagnie. Le processus de développement logiciel a été évalué au niveau un de CMM. Le groupe a établi un programme pour atteindre au moins le niveau deux de CMM. Enfin, et pour des engagements spéciaux, le projet doit être réalisé dans une période de trois mois. Le respect de la date limite fixée est fortement exigé.

V.3.3. Discussion

Facteurs du projet

Concernant le domaine d'application, nous pouvons éliminer Cleanroom qui ne représente pas un bon choix pour développer une application commerciale. Le projet est critique à la compagnie due à la date limite imposée par les dirigeants. La criticité favorise les méthodologies guidées par les plans (RUP) plutôt que les méthodologies agiles (XP et OS). Quant à la qualité, le produit n'exige pas un niveau élevé de qualité. La qualité a peu d'influence sur notre choix puisque les

trois méthodologies restantes peuvent assurer le niveau de qualité exigé. En ce qui concerne l'innovation, MDW a déjà traité plusieurs projets semblables. Le produit actuel est considéré comme une amélioration d'un ancien produit. XP traite mieux le code des anciens systèmes et donne de meilleurs résultats.

Facteurs de la méthodologie

La compagnie ne peut acquérir les outils de support exigés par RUP. Quelques outils Open Source peuvent être utilisés à la place. XP et Open Source n'exigent pas de telles ressources. Le retour sur investissement est critique au bien-être de la compagnie. Il y a un besoin pressant de recouvrir les investissements. XP offre un bénéfice à court terme tandis que RUP et Open Source promettent un bénéfice à long terme. Les responsables de MDW insistent sur l'obtention d'un niveau élevé de CMM. RUP et XP offrent plus de possibilités que Open Source. L'adoption donne plus de priorité à XP et à un moindre degré RUP. La bonne communication entre les membres de l'équipe et l'environnement ouvert à l'intérieur de la compagnie donnent plus d'acceptation à XP et facilite son adoption. RUP est rigide et exige beaucoup de ressources tandis que Open Source exige beaucoup de temps pour son adoption et a peu d'acceptation au sein de la compagnie en la considérant comme un processus chaotique.

Facteurs de l'organisation

Une équipe de sept membres est une équipe idéale pour un projet XP et elle s'adapte bien pour le développement du produit actuel. Open Source est également possible mais nous n'avons aucun indicateur quant au rendement de l'équipe. Une petite version de RUP est également possible après une opération d'adaptation. Le dynamisme favorise fortement XP et Open Source dans le cas de MDW. Le petit budget assigné au projet favorise XP et Open Source et rend l'accomplissement du projet avec RUP incertain.

V.3.4. Conclusion

L'achèvement d'un niveau élevé de maturité est un but stratégique pour MDW. Ceci servira dans l'avenir comme un élément de marketing pour rester compétitive sur le marché et pour gagner des contrats. D'autre part, il y a un besoin pressant de récupérer les investissements aussitôt que possible. Ce sont les deux objectifs principaux que la compagnie veut atteindre. Le premier objectif se coïncide avec la perspective client représentée par les deux facteurs : la qualité du

produit et la maturité du processus de développement logiciel, tandis que le second s’attache à la perspective finance.

Open Source ne satisfait pas la première expectation et n'est pas donc recommandée. Concernant l’aspect finance, le besoin d’une courte période de recouvrement de l'investissement et le petit budget assigné pour le projet font de XP un meilleur choix. Quant aux possibilités d'amélioration du processus de développement logiciel, XP et RUP offrent des niveaux de maturité élevés ce qui conforme avec les objectifs fixés. Le reste des facteurs (adoption et personnalisation, innovation, domaine d’application, qualité, criticité, et taille de l'équipe) influencent peu notre choix parce qu'ils sont traités presque de la même manière par les deux méthodologies: XP et RUP. Les facteurs précédents font de XP le choix le plus approprié au développement du projet étudié.

CONCLUSION GÉNÉRALE

Cette étude traite la comparaison et la sélection des méthodologies de développement logiciel. Le besoin de comparer les méthodologies de développement logiciel provient de la perplexité qui oppose les compagnies et les chefs de projets en choisissant une méthodologie pour leurs projets. Nous avons présenté une nouvelle approche pour la sélection basée sur le Balanced Scorecard, un modèle industriel qui a prouvé son conformité au cours du temps.

Nous avons présenté d'abord une comparaison de quatre méthodologies bien connues, en utilisant différents aspects du développement logiciel: Rational Unified Process, Extreme Programming, Cleanroom Software Engineering et Open Source Development. Ces aspects utilisés sont liés soit au projet, soit à l'organisation ou bien à la méthodologie elle-même. Nous avons présenté la qualité demandée, l'innovation, le domaine d’application et la criticité pour le projet; les outils de support, le retour sur investissement de l'amélioration du processus de développement logiciel, la possibilité de l’amélioration de la maturité du processus de développement logiciel et l'adoption pour la méthodologie; la taille de l'équipe, la discipline et les ressources disponibles pour l'organisation.

La comparaison nous a servi pour évaluer la performance des quatre approches de développement logiciel en utilisant le modèle BSC. Le BSC définit quatre perspectives complémentaires qui peuvent évaluer la capacité d'une compagnie: processus internes, finance, apprentissage organisationnel et client. Ces quatre perspectives équilibrent les résultats financiers à court terme et les possibilités concurrentielles à long terme.

Les résultats de la comparaison ont été employés pour faire concorder le projet avec la méthodologie qui répond mieux aux priorités de la compagnie. Les priorités sont représentées par les quatre perspectives du BSC. Les facteurs étudiés ont été triés autour des quatre perspectives. Chaque facteur a été lié à la perspective qu'il contribue plus à sa réalisation. La compagnie peut demander une réalisation partielle ou complète des quatre perspectives. La compagnie devrait se concentrer alors sur les facteurs liés aux perspectives désirées.

En plus des recommandations fournies par le modèle proposé pour la sélection de méthodologies, l'intérêt du modèle se trouve dans le fait qu'il prend en compte les futurs développements et projets de la compagnie et ne limite pas la vision sur un seul projet. Les méthodologies pour les futurs projets seront choisies de la même manière, assurant ainsi une continuité pour atteindre les objectifs désirés. L'environnement créé par chaque projet complétera ce qui a été déjà fait.

Cependant, ce modèle peut être amélioré en introduisant des perspectives additionnelles dans le modèle BSC pour mieux convenir au développement logiciel. En fait, le développement logiciel a ces caractéristiques spéciales qui le rendent différent par rapport aux projets industriels. Une étude approfondie sera nécessaire pour découvrir les autres perspectives possibles.

Le framework utilisé, composé de différentes caractéristiques du projet, de l'organisation et de la méthodologie n'est pas exhaustif mais montre seulement les facteurs les plus importants de ces trois éléments. L'utilisation du modèle BSC comme point de départ va nous permettre d'identifier d'autres caractéristiques appropriées et donc d'enrichir le framework avec plus de facteurs qui peuvent affecter le choix. Ceci est également valable pour les méthodologies étudiées. Introduire plus de méthodologies apportera plus de diversité et donne plus de crédibilité à l'étude.

Le Balanced Scorecard est l'un des divers modèles industriels qui ont été adoptés avec succès depuis des décennies. Le développement logiciel a des caractéristiques communes avec la communauté industrielle et a déjà emprunté beaucoup de techniques et d'outils relevant à l'industrie. Afin de réaliser une meilleure performance et résoudre la crise courante du logiciel, la communauté informatique peut rechercher des modèles ou des normes industrielles à utiliser pour améliorer la façon de développer le logiciel.

REFERENCES

[1] – Brooks P. Frederick *"No Silver Bullet: Essence and Accidents of Software Engineering"* IEEE Computer, Vol. 20, No. 4 - April 1987

[2] - B. Henderson-Sellers, G. Collins, R. Dué and I. Graham *"A Qualitative Comparison of two Processes for Object-Oriented Software Development"* Information and Software Technology, 43 (2001) 12

[3] – Gary Pollice *"Using the RUP for Small Projects: Expanding Upon eXtreme Programming"* Rational Software Whitepaper. Available at: *http://www.ibm.com/software/rational/info/literature/whitepapers.jsp*

[4] – Jason Charvat *"Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies and Processes for Projects*" John Wiley and Sons - 2003

[5] – Alistair Cockburn *"Agile Software Development – The Cooperative Game"* Cockburn and Highsmith Series - 2001

[6] – Alfonso Fuggetta *"Software Process : A Roadmap"* Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 2000

[7] – Robert C. Glass *"Process Diversity and Computing Old Wives'/Husbands' Tale"* IEEE Software, July 2000

[8] – Mikael Lindvall and Ioana Rus *"Process Diversity in Software Development"* IEEE Software, July/August 2000

[9] – Philippe Kruchten *"The Rational Unified Process: An Introduction"* Addison Wesley – 2003

[10] - Per Kroll and Philippe Kruchten *"The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP"* Addison Wesley - 2003

[11] - Pekka Abrahasson, Outi Salo, Jussi Ronkainen and Juhani Warsta *"Agile Software Development Methods – Review and Analysis"* PHD – VTT Electronics - 2002

[12] - Barry Boehm and Richard Turner *"Balancing Agility and Discipline: A Guide for the Perplexed"* Addison Wesley – 2003

[13] - Richard C. Linger and Carmen J. Trammell *"Cleanroom Software Engineering Reference Model - Version 1.0"* Technical Report CMU/SEI-96-TR-022 ESC-TR-96-022 - November 1996

[14] - R. Linger *"Cleanroom Process Model"* IEEE Software, March 1994

[15] - Robert S. Oshana and Richard C. Linger *"Capability Maturity Model for Software Development Using Cleanroom Software Engineering"* Proceedings of the 32nd Hawaii International Conference on System Sciences - 1999

[16] - Carman J. Trammell and Mark G. Ploszkoch *"The Incremental Development Process in Cleanroom Software Engineering"* Decision Support Systems - 1996

[17] – *http://www.enterpriseunifiedprocess.com/*

[18] – M. D. Deck *"Cleanroom Software Engineering and 'Old Code' -- Overcoming Process Improvement Barriers"* Proceeding of the Pacific Northwest Software Quality Conference, October, 1995

[19] - Alfonso Fuggetta *"Open Source Software—An Evaluation"* Journal of Systems and Software, Volume 66, Issue 1, April 2003

[20] - Juhani Warstaa and Pekka Abrahamsson *"Is Open Source Software Development Essentially an Agile Method?"* 3rd Workshop on Open Source, International Conference on Software Engineering, Portland, Oregon - May 3-11, 2003

[21] – Lisandra V. Manzony and Roberto T. Price *"Identifying Extensions Required by RUP to Comply with CMM Levels 2 and 3"* IEEE Transactions on Software Engineering, February - 2003

[22] - Richard C. Linger, Mark C. Paulk and Carmen J. Trammel *"Cleanroom Software Engineering Implementation of the Capability Maturity Model for Software"* Software Engineering Institute, December - 1996

[23] - *"Open Source Maturity Model"* Available at:
http://www.navicasoft.com/pages/osmm.

[24] – W. Bleek, M. Finck and B. Pape *"Towards an Open Source Development Process –Evaluating the Migration to an Open Source Project by Means of the Capability Maturity Model"* Proceedings of the First International Conference on Open Source Systems (OSS 2005), 11-15 July 2005, Genua, Italy

[25] P. Hausler, R. Linger and C. Trammell *"Adopting Cleanroom Software Engineering with a Phased Approach"* IBM Systems Journal - 33[1] 1994

[26] - Kent Benck *"Extreme Programming Explained - Embrace Change"* Addison Wesley – 1999