UNIVERSITÉ DU QUÉBEC À MONTRÉAL

LES MÉTRIQUES APPLIQUÉES DANS LA CONSTRUCTION DE LOGICIEL

MÉMOIRE

PRÉSENTÉ COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

HAO WANG

MAI 2007

REMERCIEMENTS

En premier lieu, je voudrais remercier profondément mon directeur de recherche, M. Philippe Gabrini, de m'avoir donné son soutien scientifique et moral, ses compétences ainsi que ses suggestions pertinentes et judicieuses durant la rédaction de mon mémoire;

Ensuite, je tiens également à remercier chaleureusement M. Roger Villemaire, directeur de maîtrise en informatique et à Mme Sylvie Bisson, assistante à la gestion de programmes des cycles supérieurs, un chaleureux remerciement pour leur aide et soutien;

Enfin, j'adresse ma gratitude à mes parents et ainsi à toute ma famille; je n'aurais pu réaliser mon mémoire sans leur amour, leur compréhension et leurs encouragements.

TABLE DES MATIÈRES

LISTE DES FIGURES vi
LISTE DES TABLEAUX viii
LISTE DES ACRONYMES ix
RÉSUMÉ xi

INTRODUCTION 1

CHAPITRE I : LA MESURE EN GÉNIE LOGICIEL DANS LE CONTEXTE DE LA GESTION DE PROJET 4
1.1 La planification du projet de construction 5
1.2 Processus de contrôle du projet de construction 5
1.3 Facteurs influençant le processus de construction de logiciels 6
1.3.1 Le facteur humain 6
1.3.2 Le facteur du processus logiciel 8
1.3.3 Les facteurs du matériel informatique 10
1.3.4 Les facteurs du produit 10
1.4 Concepts fondamentaux d'estimation du développement logiciel 10
1.4.1 Les caractéristiques liées à l'estimation logicielle 11
1.4.2 Le processus d'estimation logicielle 12
1.4.3 Revue des techniques d'estimation de la taille logicielle et des métriques appliquées 14

1.4.4 Modèles d'estimation des efforts de développement 23

1.5 Conclusion du chapitre 27

CHAPITRE II : REVUE ET ANALYSE DES MÉTRIQUES LOGICIELLES 29

2.1 Conception du processus de mesure en génie logiciel 29

2.1.1 Le cadre d'établissement de la mesure logicielle 30

2.1.2 Une conception des métriques logicielles 34

2.1.3 La classification des métriques logicielles 36

2.1.4 Les métriques appliquées dans la construction du logiciel 38

2.2 Revue des métriques logicielles 43

2.2.1 Métriques pour mesurer la dimension logicielle 43

2.2.2 Métriques de taille de l'objet logiciel 44

2.2.3 La mesure de la taille fonctionnelle logicielle 50

2.2.4 La mesure de la complexité logicielle 54

2.2.5 Les mesures dans le paradigme orienté objet 62

CHAPITRE III : MESURE DE LA PROGRESSION DE LA CONSTRUCTION DE LOGICIEL 68

3.1 Relation entre mesure de progression et contrôle du développement du projet 68

3.2 Conclusion pour la relation entre la mesure de progression et le contrôle du projet 73

3.3 Revue des mesures pour la progression de la construction 74

3.3.1 Mesurer la progression par le jalonnement 77

3.3.2 Mesurer la progression par le suivi de la taille du projet 79

3.3.3 Mesurer la progression en adaptant les métriques de processus 82

3.3.4 Mesurer la progression par la méthodologie EVM 88

3.3.5 Conclusion pour les métriques de progression 93

3.4 Les métriques sous les différents modèles de développement 94

3.4.1 Le modèle incrémentiel 96

3.4.2 Le modèle de développement en spirale 99

3.4.3 Le modèle Agile 103

3.4.4 Mesures utilisées dans le développement Agile 107

3.4.5 Conclusion 113

CONCLUSION 115

BIBLIOGRAPHIE 120

LISTE DES FIGURES

Figure 1.1 Relation entre l'interdépendance et la performance de l'équipe 8

Figure 1.2 Les moyennes de productivité dans les différents langages de programmation 9

Figure 1.3 Les incertitudes dans l'estimation logicielle 12

Figure 1.4 Le processus d'estimation d'un projet logiciel 13

Figure 1.5 Les phases dans le cycle de vie d'un système logiciel 15

Figure 2.1 Exemple d'utilisation de la méthode du délai dynamique 36

Figure 2.2 La gestion du projet conduite par les métriques 40

Figure 2.3 Statistiques des moyennes de NCLOC de chaque application 46

Figure 2.4 Exemple d'un graphe de contrôle de flux 56

Figure 2.5 Exemple de la démonstration des interrelations entre des unités fonctionnelles.. 58

Figure 3.1 Intégration du programme de mesure dans le processus de développement 69

Figure 3.2 Cadre de la phase de gestion de construction 71

Figure 3.3 Exemple de diagramme de Gantt appliquant le jalonnement 78

Figure 3.4 Le schéma référentiel d'intégration de code d'un système logiciel 80

Figure 3.5 Les schémas référentiels de quatre projets X, Y, A qui sont complétés et B au début de son développement 82

Figure 3.6 Le tableau de bord de la progression de l'activité de test logiciel 84

Figure 3.7 Les schémas référentiels d'apparitions de défaut 85

Figure 3.8 Le suivi des pourcentages de défauts de niveaux de sévérité 1 ou 2 dans les différentes mises en production 86

Figure 3.9 Le diagramme de Pareto présentant les différents types de défaut apparus le plus fréquemment dans le code source 86

Figure 3.10 Le suivi de la densité de défauts 88

Figure 3.11 Les métriques adoptées dans l'approche EVM 89

Figure 3.12 Exemple d'utilisation des mesures SV et CV dans un projet 90

Figure 3.13 Exemple d'utilisation des mesures SPI et CPI dans un projet 91

Figure 3.14 Exemple de calcul de la valeur de mesure ES 92

Figure 3.15 a) Le modèle de développement en cascade b) Le modèle de développement incrémentiel 97

Figure 3.16 Le modèle en Spirale 100

Figure 3.17 Exemple de structure de l'organigramme d'un projet Agile 109

Figure 3.18 Exemple des Stories et de leurs poids pondérés 109

Figure 3.19 La supervision de mesure RTF illustrée 111

LISTE DES TABLEAUX

Tableau 1.1 Un exemple de matrice de jugement M..... 18

Tableau 1.2 Tableau de la comparaison..... 26

Tableau 2.1 Composants du processus de la mesure logicielle..... 31

Tableau 2.2 Démonstration de l'exemple en utilisant l'approche GQM 32

Tableau 2.3 Les cinq niveaux de maturité et leur description dans le modèle CMMI..... 33

Tableau 2.4 Comparaison entre la méthode des points de fonction et celle du nombre de lignes de code..... 52

Tableau 3.1 La comparaison entre le modèle en cascade et le modèle incrémentiel..... 98

Tableau 3.2 Tableau de comparaison entre les trois modèles de développement..... 101

Tableau 3.3 Les caractéristiques de gestion d'un projet Agile 107

LISTE DES ACRONYMES

ACWP	Actual Cost of Work Performed
BCWS	Budget Cost of Work Schedule
BCWP	Budget Cost of Work Performed
BAC	Budget At Completion
BV	Business Value
CBO	Coupling Between Objects
CMM	Capacity Maturity Model
COCOMO	Constructive Cost Model
DIT	Depth In The Tree
DSDM	Dynamic System Development Method
EBV	Earned Business Value
EMD	Elementary Mental Discrimination
EVM	Earned Value Management
ES	Earned Schedule
FDD	Feature-Driven Development
GQM	Goal Question Metric
KLOC	Thousand Lines of Code
LOC	Lines of Codes

LCOM	Lack Of Cohesion Of Methods
NOC	Number Of Children
FP	Function Point
RFC	Reference For Class
ROI	Return Of Investments
RTF	Running Test Feature
WBS	Work Breakdown Structure
WMC	Weighted Methods per Class
XP	Extreme Programming

RÉSUMÉ

De nos jours, en raison de l'augmentation de la portée des systèmes informatiques et des exigences de la qualité de logiciel, le processus de développement de logiciel est devenu de plus en plus complexe et de plus en plus difficile à contrôler. De nombreuses métriques logicielles et des méthodes de mesure ont étés établies et utilisées comme des indices utiles et significatifs, et ce afin de mieux comprendre et maîtriser le processus de construction logicielle tout en assurant la qualité du produit final.

La présente étude est un survol des métriques logicielles et des méthodes appliquées dans la construction de logiciels ; cependant, nous classifions ces mesures variantes et ces méthodes de mesure en fonction de leurs multiples buts d'utilisation, et réalisons une étude analytique de chaque métrique en exposant ses avantages et ses inconvénients. De plus, ces analyses comparatives sont réalisées entre les différents types des métriques. Dans le présent travail, nous expliquons la complexité élevée du processus de construction en exposant les facteurs qui influencent la progression de construction logicielle. Par la suite, nous présentons et analysons les divers modèles d'estimation et les métriques appliquées dans l'estimation logicielle afin de prédire l'envergure et l'effort de développement. L'estimation du projet est considérée comme une activité indispensable dans le processus de construction. Ainsi, nous faisons une comparaison horizontale entre les modèles d'estimation d'effort de développement. Par la suite, nous effectuons une revue des métriques en mesurant la taille logicielle, la complexité de logiciel et la structure de programme orienté objet. Les analyses de ces métriques portent sur leur degré de précision de mesure et leur utilité dans la mesure de qualité logicielle. Enfin, nous mettons l'accent sur les métriques et les méthodes adoptées pour mesurer la progression de construction de logiciel ; certaines métriques et technologies de mesure sont introduites et analysées telles le jalonnement, les métriques de qualité logicielle, la technologie EVM, etc. De plus, nous faisons une étude analytique des différentes métriques impliquées dans les différents modèles de développement de logiciel.

Nous concluons qu'il est très difficile de trouver un ensemble de métriques standards dans l'application industrielle malgré un grand nombre de métriques et de méthodes de mesure qui sont inventés et adoptées, parce que la sélection et l'adaptation des métriques dépendent des différents objectifs d'utilisation, chaque objectif correspondant en effet aux métriques spécifiques les plus adaptives et les plus pertinentes.

MOTS CLÉS : métrique logicielle, progression de construction, développement de logiciel, estimation logicielle, gestion de projet, qualité logicielle, complexité de logiciel.

INTRODUCTION

Le développement de logiciel est la démarche clef du génie logiciel et la phase de construction du logiciel est l'activité cruciale de ce processus de développement, car elle consomme entre 30 et 80% du temps de production de logiciel. De nos jours, avec l'augmentation de l'envergure des projets, du temps de développement, des exigences et de la qualité du produit logiciel, la construction de logiciel devient extrêmement complexe et très difficile à maîtriser. La progression de la construction et la productivité de l'équipe sont limitées par de nombreux facteurs négatifs comme la mauvaise communication au sein de l'équipe, la complexité des tâches à faire, les changements intervenus au cours de la construction, etc. Le dépassement des coûts et le retard des échéances de livraison du produit sont des problèmes souvent rencontrés au cours du développement de logiciel. De plus, les objectifs qualitatifs en termes de fiabilité, de convivialité, d'extensibilité et de facilité de maintenance du produit final ne sont pas toujours conformes aux exigences des clients. Les développeurs et les gestionnaires doivent s'efforcer d'agir et de calibrer adéquatement dans le but de perfectionner le processus de construction en réduisant les coûts et le temps de développement et en atteignant ainsi les objectifs qualitatifs du produit final.

Les mesures logicielles jouent un rôle important dans le développement de logiciel pour mieux gérer le processus de construction tout en contrôlant les coûts de production, en assurant la qualité du logiciel. Elles sont utilisées d'une part dans la planification du projet en permettant d'estimer le projet en termes d'effort de développement, de temps prévu et d'objectifs qualitatifs prédéfinis. D'autre part, les mesures servent d'indices utiles et significatifs pour refléter et interpréter la progression de la construction, la qualité des artefacts produits et la performance de l'équipe de développement, etc. Dans un contexte de gestion de projet, les mesures appliquées dans la construction fournissent une meilleure visibilité du processus de construction; elles aident les gestionnaires sur le plan quantitatif à

mieux comprendre le statut réel du projet et à porter des jugements décisionnels de manière efficace et pertinente.

Pendant la trentaine d'années passée, diverses métriques et méthodologies de mesure de logiciel ont été proposées par les chercheurs afin de mieux aider au contrôle du projet de construction. L'objectif de notre travail consiste à classifier et à faire un revue des métriques adoptées dans la construction; nous évaluons également ces métriques de façon comparative selon les aspects de leur applicabilité, utilité, précision des mesures en nous basant sur les expériences faites par les chercheurs et les gens de l'industrie. Nous mettons l'accent plus particulièrement sur les métriques ou les méthodes utilisées qui ont pour but de mesurer la progression du développement. Puisque le domaine de la mesure du logiciel est encore peu mûr, l'utilité et l'adaptabilité des métriques proposées par les chercheurs devront être validées dans la pratique industrielle future.

Notre mémoire est organisé de la façon suivante :

- Dans le premier chapitre, nous procédons à l'analyse en décrivant le rôle de l'estimation logicielle dans le contrôle de la construction. Nous assemblons et exposons les facteurs qui peuvent apporter des imprécisions et des incertitudes durant la construction; ensuite, nous présentons et analysons les modèles et les métriques adoptés dans l'estimation de la taille des projets et de l'effort de développement du projet. Nous faisons une comparaison horizontale entre ces différents modèles d'estimation de l'effort de développement.

- Dans le deuxième chapitre, nous commençons par introduire un cadre d'établissement des mesures du logiciel proposé par Fenton; nous classifions et présentons les mesures impliquées ainsi que leur utilité durant la construction de logiciel. Enfin, nous présentons et analysons les métriques les plus adoptées dans le domaine industriel; ce sont les métriques qui mesurent la dimension du logiciel, la complexité de l'architecture dans la conception du logiciel et les programmes orientés objets.

- Dans le troisième chapitre, nous présentons les diverses méthodologies et métriques de processus qui tentent de piloter le projet. Ces mesures sont employées pour refléter l'efficacité du processus de développement et l'avancement du projet en termes de coûts, de temps prévu et des attributs qualitatifs de l'artefact logiciel. Nous faisons des études comparatives entre les différents modèles de développement en génie logiciel, tout en citant leurs propres caractéristiques dans un contexte de gestion de projet. Enfin, nous présentons les métriques pertinentes et adéquates pour chaque différent modèle de développement.

Nous terminons le mémoire par une conclusion générale, faisant une brève synthèse de notre étude sur ces variantes de métriques, et en nous basant également sur les études faites, nous présentons nos vues concernant les perspectives futures des mesures logicielles appliquées au développement.

CHAPITRE I

LA MESURE EN GÉNIE LOGICIEL DANS LE CONTEXTE DE LA GESTION DE PROJET

La mesure est définie comme un processus permettant de décrire les entités du monde réel en fonction de critères prédéfinis [30] : on assigne des chiffres ou des symboles aux attributs des entités. La mesure en génie logiciel est plutôt décrite en fonction de ses utilisations principales dans le cadre de la gestion de projet [28]:

- Pour la prédiction : il s'agit de planifier les ressources et le temps exigés pour un projet logiciel, ainsi la mesure doit estimer le projet en fonction de sa taille, de sa qualité et de son coût;
- Pour l'évaluation : la mesure permet de contrôler la progression du développement du projet et d'évaluer le produit final en employant des métriques logicielles.

La métrique logicielle permet de fournir un support quantitatif pour effectuer des mesures en génie logiciel [28]; elle est définie initialement comme une mesure de la portée d'un produit logiciel, ce qui expose certaines caractéristiques ou propriétés [29]. Dans ce mémoire, nous concentrerons nos efforts dans l'étude et l'analyse des métriques dans le domaine du génie logiciel; nous mettons l'accent particulièrement sur des métriques employées dans la mesure de la progression de la construction de logiciels.

Pour mieux illustrer cette dernière, nous faisons tout d'abord une introduction de la notation de gestion de projet : elle est une série d'activités ayant pour objectif de mener et d'assurer l'efficacité et la qualité du processus de construction [4]. Les activités qui entrent dans la

gestion de projet et qui sont impliquées dans la mesure de la progression en construction de logiciels sont mentionnées dans les sections 1.1 et 1.2.

1.1 La planification du projet de construction

Elle comprend les activités suivantes [4]:

- Définition de la portée du projet en identifiant les objectifs et les exigences du développement de projet;
- Choix du modèle de développement et délimitation du travail de chacun à travers une structure hiérarchique;
- Estimation du projet selon les coûts et le planning de développement;
- Évaluation de risques : identifier et analyser les risques possibles pendant la construction du logiciel. Constitution d'une liste de risques selon leurs impacts sur la construction du projet.

On mentionne que l'estimation du projet permet d'offrir les valeurs planifiées [4], ces dernières concernent la taille du produit logiciel final et le coût de développement, etc. Ces valeurs seront par la suite comparées avec les valeurs obtenues durant la construction de logiciels. Les techniques d'estimation et les métriques appliquées dans le processus d'estimation sont détaillées dans la section 1.4.

1.2 Processus de contrôle du projet de construction

Le processus de contrôle fournit à la gestionnaire de projet le moyen de surveiller et de contrôler la construction du logiciel [4], il utilise une série des techniques de gestion de projet. Lors du projet de construction, ces techniques apportent à la gestionnaire de projet les informations sur la progression de la construction. Cette dernière peut en déduire les déviations en comparant les valeurs planifiées et les valeurs obtenues, puis elle procède à une analyse en profondeur afin d'identifier et de résoudre les problèmes, améliorant ainsi la qualité du processus de construction en prenant les dispositions adéquates.

À partir des publications [4; 28; 29], pour ce qui suit, nous établissons et déduisons les points de vue exposés ci-dessous:

- La mesure de la progression de la construction de logiciels joue un rôle majeur dans le cadre de la gestion de projet; elle est considérée comme étant un indicateur offrant

les connaissances utiles sur l'avancement du projet de construction, aidant à mieux comprendre et maîtriser le processus de construction de logiciels;

- Les métriques donnent des mesures utilisées dans la gestion de projet; la progression de la construction de logiciels est quantifiée par les métriques, ces dernières étant caractérisées par trois attributs principaux: le coût de développement, la planification du projet et la taille du produit logiciel.
- Le processus d'estimation du projet représente une fonction principale dans la mesure de la progression en construction de logiciels; il fournit un jalon à l'équipe de développement afin d'aider la gestionnaire de projet à vérifier la conformité de l'avancement de la construction de logiciels au plan estimé. Cependant, en se basant sur le plan estimé, la gestionnaire de projet peut détecter les déviations, tout en déterminant les facteurs qui affectent la progression de la construction de logiciels.

1.3 Facteurs influençant le processus de construction de logiciels

La construction de logiciel est un processus complexe et comporte un fort degré de variabilité [31]. Afin de mener à bien la gestion de la construction de logiciels, il faut tenir compte d'un nombre important de paramètres dont les contraintes en ressources matérielles, les langages de programmation utilisés et le facteur humain. Ces derniers influencent considérablement la progression de la construction de logiciels et amènent des difficultés pour mesurer et contrôler le processus de développement de logiciels.

Dans cette section, nous faisons une revue de divers facteurs qui influencent la progression de la construction de logiciels. Ces facteurs sont perçus et constatés aussi bien par les académiciens que par les praticiens de l'industrie. Compte tenu de l'importance de ces facteurs, nous les exposons, classés dans quatre catégories [31] :

1.3.1 Le facteur humain

Il est considéré comme étant le facteur crucial affectant la performance du développement de logiciels [31]. Il peut comprendre la capacité et l'expérience des programmeurs, l'expérience de l'équipe, la taille de l'équipe de développement, l'organisation de l'équipe de développement et la qualité de la gestion de l'équipe. En nous appuyant sur [7; 8; 10; 31], nous obtenons les constatations suivantes.

- La communication entre les membres de l'équipe de développement joue un rôle positif dans le développement de logiciels. Elle permet de réduire l'ambiguïté[1] dans l'équipe en clarifiant les responsabilités de chacun dans l'équipe, ceci améliore la performance de l'équipe au cours du développement de logiciels. D'autre part, le management des membres de l'équipe permet aussi de restreindre l'ambiguïté et l'incertitude durant le développement, il comprend deux stratégies de contrôle de l'équipe :

    - La stratégie du contrôle directorial par la gestionnaire de développement qui prend en charge l'allocation de tâches aux membres de l'équipe et la supervision et l'évaluation du processus de développement de logiciels [7].

    - La stratégie du contrôle individuel est telle que les membres de l'équipe de développement assignent les tâches, et déterminent les méthodes de développements eux-mêmes [7]. Ceci réduit ou élimine les conflits au sein de l'équipe, encourage la communication parmi les membres de l'équipe, augmente la motivation dans le travail et améliore la performance de l'équipe [8].

- La structuration de l'équipe a un impact sur la performance de l'équipe de développement, cette activité possède deux caractéristiques principales [8]:

    - L'interdépendance, qui définit un domaine de coordination interactif parmi les membres de l'équipe de développement [8]. Stewart et Barrick [8] ont établi la relation entre l'interdépendance et la performance de l'équipe présenté dans la figure 1.1, on constate que le niveau extrême [2] d'interdépendance correspond à une meilleure performance de l'équipe dans la phase de conception (démontré par la courbe en trait plein dans la figure 1.1), car il symbolise une bonne communication et moins de conflits dans l'équipe. Au contraire, dans la phase de construction dans laquelle on doit avoir plus de tâches touchant le comportement (démontré par la courbe en pointillés dans la figure 1.1), le niveau modéré d'interdépendance résulte en une meilleure performance de l'équipe.

---

[1] Ambiguïté en termes de l'allocation de tâches aux membres de l'équipe de développement

[2] Le niveau extrême indique le bas niveau ou le haut niveau

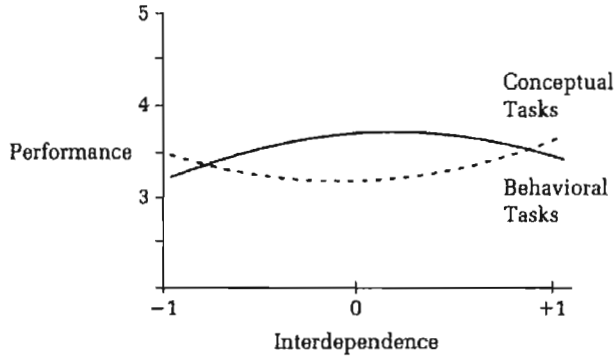

Figure 1.1 Relation entre l'interdépendance et la performance de l'équipe par Stewart et Barrick [8]

- Le contrôle individuel implique que l'équipe est libre de rediriger le développement de logiciels sans une supervision externe. Stewart et Barrick ont conclu que le contrôle individuel produit une relation linéaire et négative avec la performance de l'équipe dans la phase de construction de logiciels [8].

- Des études empiriques ont indiqué que la taille de l'équipe de développement et l'expérience de l'équipe doivent être considérées et prises en compte comme étant les deux facteurs principaux qui affectent la productivité[3] de l'équipe de développement. On cite les constatations ci-dessous :

  - Si la taille de l'équipe de développement augmente, l'équipe de développement exigera plus de coordination parmi ses membres. Ceci induit une surcharge d'effort consacrée à la construction du code, car la participation de nouveaux membres exigera du temps et des efforts à l'équipe afin d'intégrer ces nouveaux dans les travaux déjà faits.

  - L'expérience de l'équipe a un impact significatif sur la productivité de l'équipe durant la construction de logiciels. Les programmeurs avec plus d'expérience sont deux à quatre fois plus productifs que ceux avec moins d'expérience.

1.3.2 Le facteur du processus logiciel

Il comprend un ensemble de divers facteurs comme le langage de programmation, le choix du modèle de développement, le planning du développement, la disponibilité de la base de

[3] La productivité est définie comme S/E, où S représente le nombre de lignes de code produit et E représente l'effort en « homme * mois » qui est dépensé durant la construction du code.

données, les outils de développement, le langage de spécification et la conception descendante, etc.[31].

- Prechelt [11] évoque une étude comparative entre les langages conventionnels[4] et les langages de « scripts[5] »; il constate que les langages conventionnels produisent plus de code et une durée de construction deux fois plus longue que les langages « scripts » .

- Un langage de haut niveau ramène une réduction significative à l'effort du développement et correspond à une plus grande productivité [10]. La figure 1.2 indique la productivité en fonction des langages de différents niveaux [10].

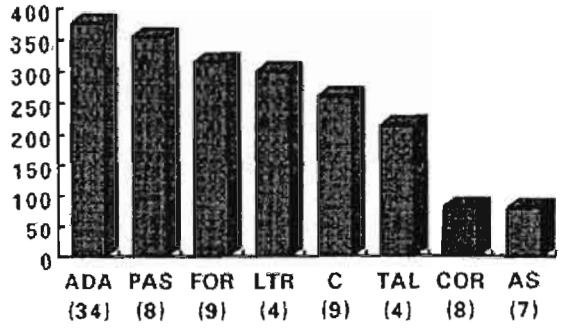

Figure 1.2 Les moyennes de productivité dans les différents langages de programmation par Maxwell, Wassenhove et Dutta [10]

- À partir de la figure 1.2, le langage Ada présente la productivité la plus haute, parce que les projets logiciels en Ada présentent moins d'exigences en termes d'allocation de mémoire, de fiabilité et du temps moyen de réponse, de plus ils permettent une meilleure utilisation de l'outil de programmation et des pratiques de programmation [10]. Cela se traduit par une réduction significative de l'effort de développement. C'est le cas contraire dans les projets en langages de bas niveau tels Coral et Assembleur qui ont une productivité plus basse.

- Molokken et Jorgensen procèdent à une analyse comparative des modèles de développement [6]. Ils observent que le modèle incrémentiel et le modèle évolutionnaire introduisent moins de surcharge d'effort de développement que le modèle en cascade.

---

[4] Les langages conventionnels exigent des déclarations de types de variables afin d'être compilés comme C, C++, Java.

[5] Les langages de scripts sont interprétables plutôt que compilés comme Perl, Python, Rexx et TCL.

1.3.3 Les facteurs du matériel informatique

Les contraintes en ressources matérielles informatiques peuvent avoir des impacts négatifs sur le développement de logiciels [31]. Ils peuvent comprendre le temps de réponse, le temps d'exécution ou les contraintes en mémoire, le matériel informatique dans le cas de développement simultané, la volatilité du système.

1.3.4 Les facteurs du produit

Les facteurs liés au produit ont une influence sur les logiciels produits [31]. Ces facteurs comprennent la taille du produit, les exigences en temps réel, les attributs de la qualité du logiciel tels que les structures de données, la fiabilité, la portabilité, les structures de contrôle, le nombre de modules et le couplage entre les modules, la complexité du logiciel, la quantité de réutilisation du code, la quantité de documentation du code, les contraintes en sécurité du logiciel et le type de logiciel.

- La réutilisation logicielle se définit comme la réutilisation de produits logiciels existants, sans y apporter des modifications significatives, afin de les intégrer dans d'autres constructions logicielles [12]. Elle améliore la qualité du processus de développement et la productivité de l'équipe. En d'autres termes, la réutilisation signifie moins de nouveau code et réduit l'effort de développement; elle procure un gain substantiel sur le nombre de développeurs et la durée nécessaire au développement du logiciel.

- Les modifications de conception se produisent souvent au cours de la construction de logiciels; cependant, la conception est appelée à changer au cours du développement et à s'ajuster dynamiquement en fonction des changements des exigences commerciales. Ces derniers ont un impact négatif sur le développement, car ils se répercutent dans divers modules en construction et exigent une remise en question du développement ou des nouvelles fonctions liées aux modules concernés. Ceci pourra surcharger l'effort de développement et résulter en un délai supplémentaire sur la progression de la construction.

1.4 Concepts fondamentaux d'estimation du développement logiciel

Comme mentionné dans la section 1.1, l'estimation du logiciel est l'une des plus importantes activités de la planification du projet de développement de logiciel, et est définie comme un processus de prédiction de la durée et du coût du projet logiciel [14]. Ce processus est appliqué avant de commencer la phase de construction du projet, afin de prédéterminer le

temps à prévoir, le budget à obtenir et les fonctions du logiciel à réaliser afin de mener à bien le développement du logiciel. L'estimation du coût d'un projet logiciel met en jeu les deux attributs qui sont strictement liés: l'effort de développement et le temps nécessaire au développement. Le terme « effort de développement » indique le nombre de personnes et le nombre de mois nécessaires afin d'achever un projet logiciel, c'est-à-dire qu'il donne une mesure en «Homme-Mois »; en conséquence, cet effort pourra être converti en coût monétaire (en dollars) si on connaît les salaires des personnes dans le projet; on calcule le coût du développement en multipliant ces salaires mensuels par le nombre de mois du développement [18]. On constate que l'estimation du coût de développement d'un projet logiciel est centrée sur l'effort humain, et mesure le projet en terme de « homme-mois ».

1.4.1 Les caractéristiques liées à l'estimation logicielle

Comme le développement du projet logiciel est un processus complexe, si l'on considère les facteurs d'incertitude cités dans la section 1.3, ceux-ci entraînent des inexactitudes dans l'estimation logicielle. Selon Sommerville [32], il est difficile de procurer une estimation rigoureuse de l'effort de développement dans les phases initiales de développement de logiciel, parce que l'estimation logicielle se base sur la spécification des besoins de la clientèle à un haut niveau; celle-ci comprend des ambiguïtés, des confusions et l'estimation est souvent modifiée au cours de la construction à cause des facteurs humains et des contraintes en ressources matérielles.

McConnell a conclu que l'estimation logicielle est un processus de raffinement progressif [33]; cela veut dire que la taille de l'incertitude en estimation logicielle sera diminuée et minimisée progressivement au fur et à mesure de l'avancement du développement de logiciel. Dans la figure 1.3, McConnell représente ce fait et il constate que, théoriquement, il est impossible d'avoir une estimation précise dans les phases initiales du développement de logiciel, comme la phase d'analyse de faisabilité, la phase d'analyse des besoins, et la phase de conception.

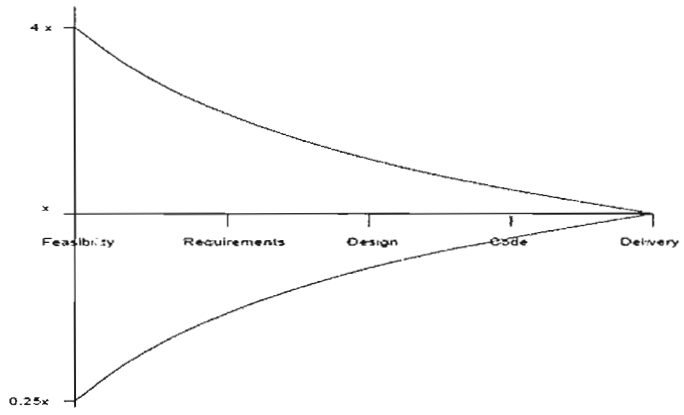

Figure 1.3 Les incertitudes dans l'estimation logicielle par McConnell [33] en adaptant le cycle de vie d'un système logiciel [32]

1.4.2 Le processus d'estimation logicielle

À partir de la figure 1.3, on voit que la sous-estimation ou la surestimation de la taille du logiciel est quatre fois celle du produit final dans la phase d'analyse de faisabilité; par la suite, cette incertitude est diminuée jusqu'à 25% dans la phase de conception. On constate que l'estimation logicielle est un processus continuel durant tout le cycle de vie d'un système logiciel; elle évalue et ajuste les valeurs d'estimation de manière dynamique afin d'atteindre une précision plus acceptable lors du développement du projet logiciel [14]. L'établissement d'un tel processus d'estimation au commencement[6] du cycle de vie d'un projet logiciel permet d'avoir une idée plus fidèle [16]; cela peut aider à identifier et à saisir les facteurs qui affectent l'effort et le temps de développement du projet assez tôt; cependant, le processus d'estimation comprend des méthodes pour identifier et suivre ces facteurs de risque, qui peuvent résulter en une surcharge de coût et de temps du développement logiciel. Le processus d'estimation d'un projet logiciel est illustré par la figure 1.4.

---

[6] Le commencement du cycle de vie d'un projet logiciel comprend les phases initiales telles que la phase d'analyse de faisabilité, la phase d'analyse des besoins et la phase de conception.

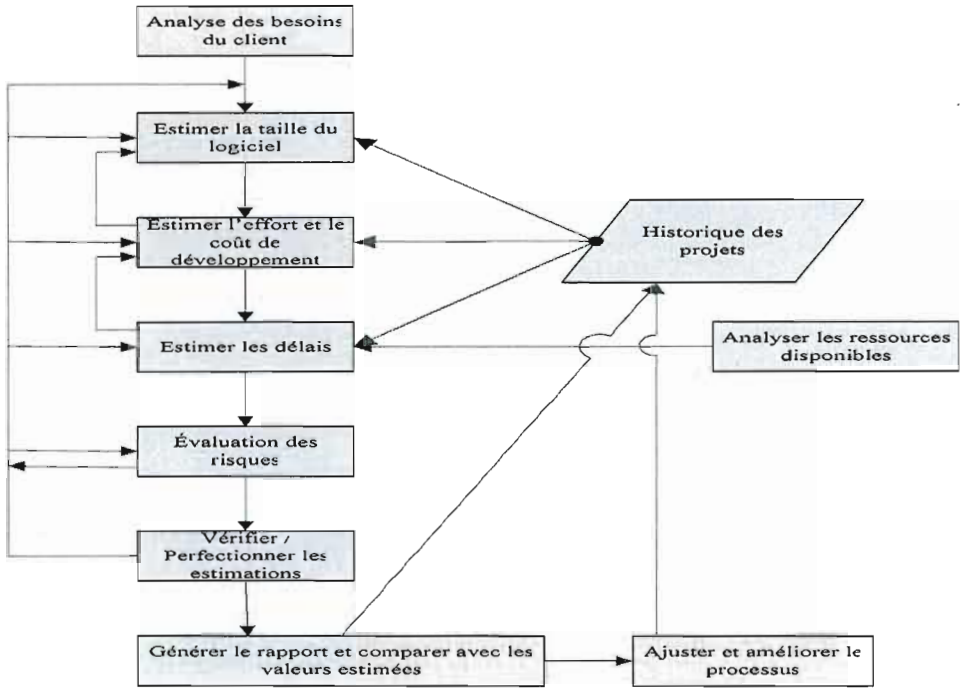

Figure 1.4 Le processus d'estimation d'un projet logiciel par Agarwal, Kumar, Mallick, Bharadwaj et Anantwar [16]

Selon la figure 1.4, le processus d'estimation comporte les activités cruciales qui suivent :

- Estimation de la taille du logiciel à développer en fonction des informations préliminaires [15]; ces dernières comprennent les spécifications formelles du projet comme la spécification des besoins du client ou les exigences du projet logiciel;

- Estimation de l'effort et des coûts de développement du projet après avoir fait l'estimation de la taille du produit logiciel [15], l'estimation de l'effort consiste à identifier et à évaluer la charge des travaux tels la documentation d'analyse, la réalisation des prototypes du logiciel, la conception du logiciel, la construction du logiciel, le test du produit logiciel; tout ceci est nécessaire pour achever le projet logiciel et les mesures sont en unités «homme-mois ». Divers modèles algorithmiques ont été conçus en paramétrant la taille estimée du logiciel afin d'estimer l'effort de développement du logiciel. Par la suite, les coûts de développement seront obtenus simplement en multipliant l'effort estimé par le salaire mensuel de chaque personne;

- Estimation des temps en considérant les ressources disponibles; il s'agit d'identifier les composants affectés au projet afin de connaître les tâches exigées

dans le projet [14; 15]; ces dernières seront représentées dans une structure hiérarchique établie par la stratégie de l'organigramme des tâches[7], qui permet d'ordonnancer ces taches et ainsi de réaliser le planning du développement tout en prédéterminant leurs temps de développement.

1.4.3 Revue des techniques d'estimation de la taille logicielle et des métriques appliquées

L'estimation de la taille du logiciel est un processus essentiel et important dans l'estimation d'un système logiciel, dont elle est la première activité de prédiction. Elle a des impacts directs sur l'exactitude de l'estimation des efforts de développement, parce que les résultats de l'estimation de la taille du logiciel sont utilisés comme paramètres d'entrée essentiels dans la plupart des modèles d'estimation de l'effort de développement [18]. D'après Laranjeira [21] et Lokan [22], durant le cycle de vie d'un système logiciel illustré par la figure1.5, le procédé d'estimation de l'effort de développement dans les phases initiales[8] a une plus grande utilité que celui de la phase de conception détaillée et de la phase de construction pour mener à bien la gestion et le contrôle du développement de logiciel; en conséquence, l'estimation de la taille du logiciel basée sur les descriptions formelles du système logiciel dans les phases initiales a des effets positifs sur le processus de développement de logiciel. De ce fait, il est nécessaire et significatif d'effectuer une estimation de la taille dans les phases initiales la plus précise possible; en pratique, il est cependant difficile d'arriver à une estimation de la taille précise dans les phases initiales [21], parce qu'elle demande des connaissances relatives à la frontière du projet suffisamment spécifiées et complètes. Ces connaissances concernent les spécifications des fonctions du système logiciel en termes de leur portée, de leur complexité et des interrelations entre ces fonctions; ces dernières sont difficiles à préciser au cours des phases initiales, dans lesquelles il demeure souvent des ambiguïtés et des confusions.

---

[7] Organigramme des tâches correspond à «WBS » ou « Work Breakdown Structure » en anglais

[8] Les phases initiales comprennent la phase de spécification et la phase de conception fonctionnelle

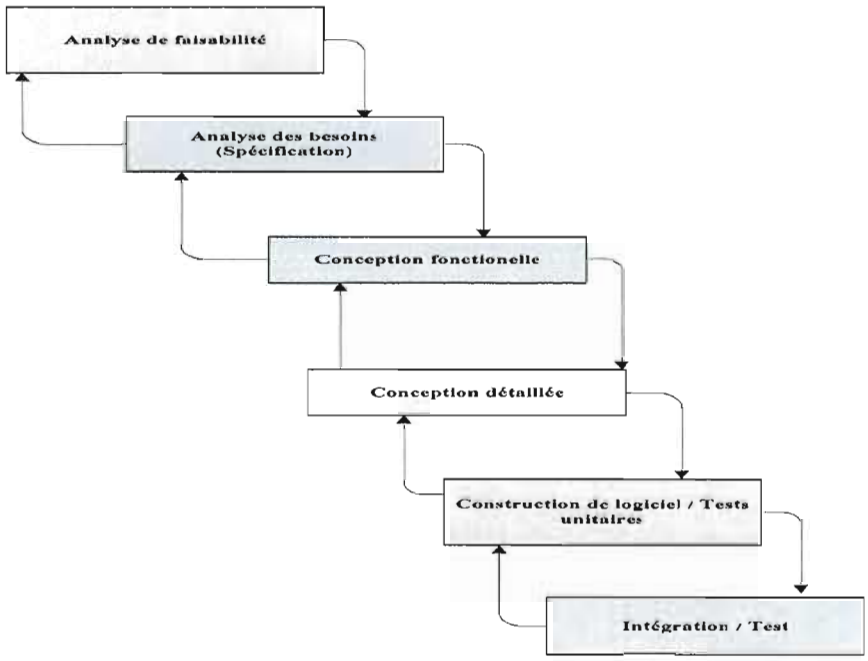

Figure 1.5 Les phases dans le cycle de vie d'un système logiciel par Laranjeira [21]

Afin de fournir des mesures quantitatives pour la gestion du développement et aussi pour l'estimation de la taille, deux unités de mesure principales sont adoptées dans les techniques d'estimation de la taille tels LOC ou KLOC[9] et FP[10] [18]. Bien qu'il existe d'autres métriques disponibles pour l'estimation de la taille comme VARS (le nombre de variables et de constantes dans un logiciel) et Token Counts (le nombre d'unités lexicales fondamentales dans un logiciel) [27;31], en pratique LOC et FP sont les deux métriques les plus utilisées dans la mesure de la taille logicielle[18]; dans cette section, nous nous concentrons sur ces

[9]LOC (Lines of Code en anglais) et KLOC (Kilo Lines Of Code en anglais) expriment le nombre de lignes de code source livrable d'un produit logiciel.

[10] FP (Function Point en anglais) exprime le nombre de fonctions d'un système logiciel, il est calculé par la méthode des points de fonction, initialement proposée par Albrecht en 1979 [17]; on présentera cette technique d'estimation dans la section 1.4.3.2.

deux métriques afin de les comparer en fonction de leur utilisation dans le processus d'estimation.

Un inventaire des diverses alternatives d'estimation de la taille a été développé; ces techniques d'estimation sont catégorisées en quatre démarches différentes [20; 21]. Celles-ci sont basées sur l'analyse par analogie [15;20;21], le modèle par jugement d'expert [20;21;26], le modèle des points de fonction [17;23;27;37], le modèle par composant [20;25;27;34;35]. Ces modèles ont chacun des avantages et des inconvénients, et on doit déterminer les adaptations de ces modèles en fonction de leur situation et de l'environnement du développement. Dans ce qui suit, nous présentons brièvement ces différents modèles d'estimation de la taille et analysons ainsi les conséquences que leur choix implique, tout en exposant les avantages et les limitations de chaque modèle d'estimation. Nous mettons ensuite l'accent sur le modèle des points de fonction en faisant une comparaison de l'application des métriques LOC et FP dans le processus d'estimation de la taille.

1.4.3.1 Analyse par analogie

La méthode d'analyse par analogie se fait à partir de projets achevés, qui possèdent des caractéristiques et des expériences de développement similaires au projet en cours de développement. L'essentiel de l'approche est de décomposer le système à développer en plusieurs modules selon les fonctions principales du système; ces dernières sont définies et spécifiées dans la phase d'analyse des besoins du client. La taille de chaque module du nouveau projet sera estimée en la considérant comme un pourcentage de la taille du module similaire du projet achevé; on estime la taille du système entier en additionnant les tailles estimées de tous les modules [15; 21], ceci est illustré par la formule suivante [20]:

$S = \sum F \times Taille$ où « S » signifie la taille estimée du système entier, « F » est un paramètre déterminé par l'expérience ou la politique d'estimation, et le paramètre « Taille » représente la taille du module similaire du projet achevé.

Analyse du modèle

L'estimation par analogie est basée sur les données historiques de chaque organisation de développement, l'avantage qu'elle offre est sa simplicité d'utilisation; mais à cause de sa dépendance aux données historiques, le modèle présente certains inconvénients comme la restriction du type de langage employé et l'environnement d'application, ainsi que la difficulté de l'appliquer dans différentes organisations [20].

1.4.3.2 Estimation par le jugement d'expert

Dans cette approche, un ou plusieurs experts sont consultés pour leur connaissances et leur expérience en estimation, afin de prédire la taille finale du système logiciel; une telle estimation est basée sur le modèle par analogie, en effet, parfois les experts infèrent la taille en se basant sur les données historiques. Ce modèle est souvent employé conjointement avec d'autres méthodes d'estimation tels que la technique PERT[11] [20; 21], la méthode de comparaison par appariement [26]. On présente ces deux méthodes dans les paragraphes suivants :

Dans la technique PERT, le système est décomposé en plusieurs modules, chaque module est estimé en utilisant l'équation d'estimation de la taille, cette dernière est suggérée par Putnam et Fizsimmons, elle est décrite comme suit :

$S_i = \frac{O_i + 4M_i + P_i}{6}$ $\qquad S = \Sigma S_i$ Le terme « S_i » indique la taille estimée de chaque module, le « O_i » est la borne inférieure de la taille estimée, c'est-à-dire « la plus optimiste », le « M_i » est la taille moyenne pondérée la plus prometteuse, le « P_i » représente la borne supérieure de la taille estimée, c'est à dire « la plus pessimiste». Évidemment, la taille estimée du système est la somme des tailles estimées de tous les modules.

[11] PERT résume l'expression anglaise « Program Evaluation and Review Technique ».

La méthode de comparaison par appariement consiste à faire une estimation de la taille à partir des tailles relatives des entités[12] du système à développer. Cette approche tend à améliorer l'exactitude de l'estimation par une procédure mathématique en fonction des tailles des entités prédites par les experts, elle comporte les étapes suivantes [26].

Établir la matrice de jugement, dont un exemple est donné dans le tableau 1.1; la matrice contient les éléments qui présentent les tailles relatives des entités du système, (ratio des tailles de deux entités du système).

Entités du projet	D	B	C	*A*
D	1	4	6	*7.5*
B		1	1.5	*2*
C			1	*2*
A				1

Tableau 1.1 Un exemple de matrice de jugement M issu de Miranda [26], chaque ligne est triée en ordre ascendant, l'élément M[1,2] est égal à 4, ceci indique que l'entité D est quatre fois plus grande que l'entité B, etc.

En fonction de la matrice de jugement, un vecteur nommé « échelle de rapport » sera construit en utilisant la procédure de la moyenne géométrique proposée par Crawford et Williams. Elle calcule la moyenne géométrique par la formule suivante :

[12] Les entités peuvent être les exigences du client, les cas d'utilisation, les modules ou les objets du système logiciel, etc. [26]

$V_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}}$ où « a_{ij} » est l'élément de la matrice de jugement, « V_i » est la moyenne géométrique calculée pour chaque entité du système. On pourra ensuite déterminer le vecteur échelle de rapport, qui est calculé par la formule suivante :

$r_i = \frac{V_i}{\sum_{k=1}^{n} V_k}$ où « r_i » est l'élément i du vecteur échelle de rapport; il exprime la proportion de la taille de l'entité i par rapport au système complet.

À la fin, les données historiques du projet précédent sont exigées afin d'obtenir la taille estimée de chaque entité. L'estimation se fait par la formule ci-dessous :

$T_i = \frac{r_i}{r_{historique}} \times T_{historique}$ où « T_i » est la taille estimée du projet, « r_i » est le rapport de l'entité i, $r_{historique}$ indique le rapport historique de l'entité correspondante du projet achevé, $T_{historique}$ est la taille de référence du projet achevé.

Analyse du modèle par jugement d'expert

Selon l'étude comparative entre la méthode de comparaison par appariement et l'estimation par jugement d'expert [26], Miranda observe que la méthode de comparaison par appariement permet d'améliorer l'exactitude de l'estimation de la taille de façon significative. L'autre amélioration due à la méthode est qu'elle définit et construit le tableau d'échelle qui définit les échelles des entités du projet. Chaque échelle correspond à une étendue de taille relative au système, et ce tableau permet d'aider les estimateurs à mieux comprendre l'échelle des entités du projet; ceci économise du temps pour l'estimation de la taille. L'approche par jugement d'expert est une estimation basée sur un jugement personnel, ce qui est souvent subjectif. Cela influe sur la sous-estimation ou la surestimation inévitable pendant l'estimation de la taille. Selon les expériences [21], Laranjeira indique que le modèle par jugement d'expert ne peut pas fournir une estimation précise à cause des raisons psychologiques et personnelles des experts; il constate qu'il y a aussi une tendance des

experts à sous-estimer la taille dans la technique PERT. Ceci est dû aux facteurs humains comme le manque d'expérience en estimation, le manque de compréhension de la conception globale du projet, le rappel incomplet des expériences précédentes, etc. De ce fait, l'estimation de la taille exige des méthodes plus performantes afin d'améliorer l'exactitude de l'estimation.

1.4.3.3 Estimation par la méthode des points de fonction

La méthode des points de fonction a été introduite par Albrecht en 1979, puis, après la formation de l'organisation IFPUG[13], cette méthode a été standardisée afin de clarifier les règles de la méthode et de promouvoir son utilisation [17]. Brièvement, la méthode tente d'estimer la taille d'un système logiciel selon des fonctions prédéfinies; ces dernières sont obtenues à partir des documents générés dans les phases préliminaires comme la phase d'analyse et la phase de conception [17;23]. La méthode des points de fonction comporte les étapes suivantes.

Tout d'abord, on identifie les composants du système en fonction des cinq types suivants :

- le type d'entrée identifie les données entrées par l'usager dans l'application,
- le type de sortie indique les données qui sortent de l'application et qui sont souvent le résultat du traitement d'une application,
- le type GDI[14] représente les groupes de données logiquement liées ou des paramètres de contrôle qui sont mis à jour et utilisés à l'intérieur d'une application du système,
- le type GDE[15] identifie les fichiers d'interfaces, qui sont mis à jour par une autre application,
- le type d'interrogation concerne les requêtes interactives qui exigent une réponse.

---

[13] IFPUG est l'organisation « International Function Point User Group »

[14] GDI est l'abréviation de « groupe logique de données internes »

[15] GDE est l'abréviation de « groupe de données externes »

Pour chaque composant du système, on détermine son niveau de complexité, classifié comme Faible, Moyen et Élevé. Chaque niveau de complexité d'un composant correspond à un poids prédéfini, qui varie de 3 à 15.

Le processus de comptage donne le nombre de points de fonction brut. Après avoir compté le nombre de points de fonction de chaque type, le calcul des points de fonction du système se fait en ajoutant les poids des différents types de composants, il est décrit par la formule suivante :

$NPF = \sum NF \times Poids$ où *NPF* est le nombre de points de fonction brut, *NF* est le nombre des fonctions dans chaque composant, *Poids* représente le poids associé à chaque type.

Le processus d'ajustement des points de fonction comporte d'abord la détermination de la valeur VAF [16], et ensuite le nombre de points de fonction estimé est ajusté par la multiplication du *NPF* et de la valeur VAF.

Analyse de la méthode de point de fonction

Cette méthode possède certains avantages [38] : d'abord la méthode est basée sur la vue externe de l'utilisateur, et est indépendante du langage de développement utilisé; ensuite, cette mesure peut être déterminée dans les phases initiales du développement, ce qui permet que la méthode s'applique au processus d'estimation; enfin, la méthode offre une simplicité de compréhension et une facilité d'application. Les limites de la méthode sont aussi évidentes [23] : l'exactitude de la méthode dans l'estimation est influencée par des facteurs comme l'expérience de la personne effectuant le compte du nombre de fonctions et son expérience en développement de logiciel. La méthode peut permettre des divergences pendant son utilisation par les différents analystes de points de fonction, à cause des jugements subjectifs de ces différents analystes comme la détermination de la complexité de chaque composant,

[16] La valeur VAF, «Value Ajustement Factor », est obtenue par une variable prédéfinie GSC, « General System Characteristics », laquelle évalue l'environnement de développement du système en ce qui concerne les réponses à 14 questions. La formule suivante permet de calculer la valeur VAF : $VAF = 0.65 + \sum C_i$ où C_i correspond au degré d'influence de chaque GSC pour le système.

l'interprétation du document de spécification, etc. À l'aide des résultats expérimentaux [23], Graham et Ross observent que la mesure des points de fonction est plus susceptible de bien estimer la taille du projet que la mesure de nombre de lignes de code dans les phases initiales, car l'estimation de la taille en utilisant le nombre de lignes de code est souvent basée sur l'expérience de l'analyste (projets achevés). Cette dernière est un processus très subjectif comportant des incertitudes, alors que la méthode des points de fonction se fait par application d'un ensemble de règles formalisées, qui résulte en une estimation plus cohérente et plus précise que la mesure du nombre de lignes de code.

1.4.3.4 Estimation par composant

Cette approche est proposée par Verner et Tate [27]; l'estimation dans cette approche exige la partition du système en plusieurs modules selon des types prédéfinis, ainsi que l'estimation de la taille des modules individuels et finalement on ajoute ces tailles estimées afin d'obtenir une taille estimée globale. Cependant, Verner et Tate divisent le système en plusieurs modules selon les types incluant le menu, les rapports, l'écran et les relations; le nombre de modules dans chaque type sera compté, ensuite ils appliquent les équations d'estimation afin d'obtenir une régression linéaire entre le nombre de modules de chaque type et le nombre de lignes de code respectives. Ce modèle d'estimation par composant a été validé par Dolado [35]; il constate que cette alternative prédit le nombre de lignes de code du système tout en quantifiant ses composants. Ceci donne une prédiction assez précise et le processus d'estimation est ainsi plus contrôlable que par la méthode des points de fonction.

L'autre alternative d'estimation est semblable à l'approche d'estimation par composant; elle est basée sur la régression linéaire, et a été développée par Ikatura et Takayanagi en 1982 [20;34]. Ce modèle d'estimation est adapté d'une étude de cas dans le développement d'un système bancaire en utilisant le langage Cobol; Ikatura et Takayanagi ont sélectionné 10 variables valables qui affecteront la taille du système donnée en nombre de lignes de code. Ces variables décrivent et partitionnent le système en nombre des modules d'entrée, en nombre des modules de sortie, en nombre des objets de chaque type de transaction bancaire dans le rapport, en nombre des objets verticaux dans le tableau à deux dimensions du rapport, en nombre des objets horizontaux dans le tableau à deux dimensions du rapport, etc. Le

modèle de régression linéaire multiple d'estimation de la taille est construit de la façon suivante :

$Y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + \beta_3 * X_3 + \ldots + \beta_{10} * X_{10}$ où Y représente le taille estimée totale, les X_i sont les variables choisies, les coefficients β_i sont déterminés et évalués par les experts d'estimation selon leur expérience des projets déjà achevés. Après avoir évalué le modèle, Ikatura et Takayanagi ont observé et conclu qu'effectivement le modèle génère un résultat d'estimation assez précis. Ce processus de modélisation est généralement adaptable aux autres types de projet de développement exigeant la sélection pertinente des variables et des valeurs des coefficients.

Analyse du modèle par composant

Verner et Tate ont constaté que l'estimation par composant offre certains avantages [27]. Ce modèle possède une intégrité d'application, c'est-à-dire peut être adapté à différents environnements de développement et fournit une taille estimée précise; il estime la taille du projet dans les phases initiales du développement; les équations de régression linéaire multiple sont établies pour obtenir la taille estimée. En conséquence, un projet achevé avec un résultat d'estimation modéré peut fournir les données suffisantes pour développer le modèle d'estimation; ceci élimine l'exigence de disponibilité de la base de données historique à grande échelle. Ce modèle présente aussi certaines difficultés pendant le processus d'estimation : manque de visibilité des composants du système dans les phases initiales, complexité dans le processus de construction qui va affecter l'exactitude de l'estimation de la taille, etc.

1.4.4 Modèles d'estimation des efforts de développement

Après avoir obtenu la taille estimée d'un système logiciel, considérée comme étant le facteur le plus significatif pour l'estimation des efforts de développement, nous pourrons en déduire l'estimation des efforts. Il existe cependant plusieurs modèles utilisés dans l'industrie; selon Agarwal, Kumar, Mallick, Bharadwaj et Anantwar [16], on peut les grouper en deux catégories principales :

- Les modèles paramétriques

  Ce genre de modèle est toujours basé sur une équation mathématique centrale[15;16;18]; une fonction f représente la relation entre l'effort de développement et les facteurs impliqués dans le développement tels que la taille du logiciel, l'expérience du personnel, la complexité du système à implémenter, le degré de réutilisation, etc. Ces facteurs, qui affectent le coût du développement, seront estimés au préalable et considérés comme des variables d'entrée principales dans la mise au point du modèle. Les techniques le plus adoptées du modèle sont listées dans ce qui suit [16;18].

  - La régression linéaire multiple : dans cette technique d'estimation d'effort, un ensemble de variables affectant le coût du développement sont adoptées dans un fonction statistique de la régression linéaire multiple, qui est de la forme suivante :

    $Effort = a_0 + a_1\chi_1 + \ldots\ldots + a_n\chi_n$ tels que les coefficients a_i sont des paramètres constants choisis et déterminés à partir des projets achevés, les variables x_i sont des facteurs qui dépendent du projet

  - Les modèles COCOMOs[17] : ces modèles ont été mis au point par Boehm initialement (COCOMO I) en 1981, la deuxième version (COCOMO II) fut mise en place en 1995. Le modèle est basé par la formule générale suivante :

    $Effort = a \times S^b$ où $Effort$ représente la charge du développement en unités de mesure « homme-mois », S est la taille estimée du logiciel représentée par le nombre de lignes de code (LOC) ou le nombre de points de fonction (FP), le paramètre a est un coefficient permettant d'introduire les 16 facteurs affectant l'effort comme la taille de la base de donnée, la fiabilité exigée du projet, la complexité du produit, l'expérience du programmeur, l'utilisation d'outils logiciels, etc. Ces facteurs seront identifiés et sélectionnés durant le processus d'estimation et à chaque facteur correspond un degré pondéré, le coefficient a est calculé par le produit de ces facteurs. Enfin, l'exposant b est déterminé par l'adoption du mode de développement du projet, ce dernier est décrit par l'environnement de développement, l'exigence de développement du projet et le degré de réutilisation du projet précédent.

- Les modèles heuristiques

---

[17] COCOMO est l'abréviation de « Constructive Cost Model »

Cette approche consiste plutôt à utiliser les données historiques des projets réalisés afin de déterminer la charge du projet actuel [16;18;39]; selon les experts en estimation et en analyse d'analogie entre le projet actuel et les projets précédents, ces techniques sont cruciales afin d'effectuer l'estimation, mais en pratique il est souvent très ardu de fournir une estimation d'effort précise en utilisant ces techniques informelles. Ceci est dû à des contraintes comme le jugement subjectif des experts, la difficulté de trouver des projets analogues et d'évaluer la similarité entre le projet actuel et des projets historiques. Nous exposons les techniques adaptées ci-dessous [16;18].

- Jugement d'expert : dans cet approche heuristique, un groupe d'experts est formé; ils procèdent à leur estimation individuellement, les résultats d'estimation d'effort seront rassemblés et évalués; l'approche est souvent employée en incorporant la technique Delphi, comportant un processus procédural. Cependant, l'équipe d'estimation doit posséder assez de connaissances des exigences du client, et chaque expert doit remplir un formulaire d'estimation, après réunions et discussions des estimateurs, chacun produit son propre résultat de la charge estimée du projet et le retourne à un évaluateur d'équipe qui va noter tous ces résultats. Une telle procédure se répète progressivement durant les phases de développement initiales avant le processus de construction du projet, et finalement, on obtient un résultat qui est la moyenne des résultats fournis dans les tours précédents.

- Heuristique ascendante et descendante : l'approche ascendante consiste à décomposer le système en plusieurs modules selon la classification des tâches à accomplir. Un arbre sera construit avec les tâches représentées par les noeuds de l'arbre : on commence à estimer les efforts nécessaires des sous-tâches au niveau le plus bas dans l'arbre et on en rassemble les résultats afin d'obtenir les efforts des tâches au niveau supérieur; on atteint ainsi progressivement le résultat estimé du système global.

  L'heuristique descendante entreprend de livrer une estimation d'effort à partir du système global, en se basant sur le jugement d'expert ou l'estimation par analogie des données historiques. Par la suite, l'effort estimé total sera réparti et assigné aux composants du système à développer.

- Analyse par analogie : dans cette technique, il faut identifier les caractéristiques du projet à estimer et trouver les projets similaires en fonction de ces caractéristiques dans la base de données historique. La consultation d'un ou plusieurs experts avec expérience est nécessaire afin d'aboutir à une estimation d'effort la plus précise possible.

Analyse et comparaison entre le modèle paramétrique et le modèle heuristique

On peut résumer les analyses comparatives entre l'approche paramétrique et l'approche heuristique de Agarwal, Kumar, Mallick, Bharadwaj et Anantwar [16] et de Leung et Fan[18] basées sur certaines études de cas, par le tableau 1.2 ci-dessous.

		Avantages	Inconvénients
Paramétrique	Régression linaire multiple Modèles COCOMO	Basé sur l'équation centrale mathématique en fournissant des résultats objectifs, répétables et contrôlables. Offre une meilleure compréhension de la méthode d'estimation.	Les paramètres d'entrée sont subjectifs dans l'équation. Modèles spécifiés et construits en fonction des circonstances comme l'environnement de développement et la compagnie, manque d'utilité en général. Équations établies et ajustées à partir des projets historiques, mais risque de mal s'adapter au projet actuel.
Heuristique	Modèle ascendant et descendant Jugement d'expert Analyse par analogie	Estimation performante en termes de son efficacité et sa rapidité. En fonction des projets similaires achevés, des opinions des experts avec expérience, on peut obtenir des résultats estimés bien précis Modèle ascendant établit l'estimation basée sur une analyse plus détaillée du système en décomposant les tâches individuelles, ce qui est plus stable et plus précis que les autres méthodes heuristiques.	Décisions subjectives inévitables durant l'estimation. Analyse d'analogie rencontre des difficultés afin d'identifier les projets semblables dans la base de donnée historique et évaluer la similarité entre ces projets. Modèle descendant estime le projet à partir d'une vue globale du système et est moins précis que les autres méthodes Modèle ascendant exige plus d'effort et plus de connaissances préalables du système.

Tableau 1.2 Tableau de la comparaison entre l'approche paramétrique et l'approche heuristique

1.5 Conclusion du chapitre

Selon les références étudiées dans le chapitre, nous établissons les points de vue et les discussions suivants.

- En résumé, on note que le développement du projet logiciel est un processus compliqué et difficile à contrôler; de nombreux facteurs produisent des effets négatifs durant le processus; la gestion de projet est nécessaire pour mener à bien le développement du projet. La mesure en génie logiciel est une démarche qui a pour but de mesurer certains attributs du processus et de donner une meilleure compréhension et un meilleur contrôle à la gestion du développement de logiciel; elle permet de prédire la taille et l'effort du projet, et d'identifier et de mesurer les facteurs affectant la progression de la construction, la qualité du produit logiciel, la productivité de l'équipe de développement et le budget du projet. Les métriques logicielles permettent d'offrir un support quantitatif dans la mesure en génie logiciel. Dans le prochain chapitre, nous allons faire une revue et une analyse des diverses métriques appliquées dans le développement du logiciel en nous concentrons sur la phase de la construction de logiciel.

- Nous avons remarqué que le facteur le plus significatif influant sur le développement du projet est le facteur humain. Cependant, la communication dans l'équipe, la capacité et l'expérience des programmeurs et la gestion de l'équipe de développement ont étés prises en compte : évidemment une équipe bien structurée aura plus de productivité et moins de délais au cours de la construction.

- Nous constatons que l'estimation logicielle est une étape nécessaire et difficile dans les phases initiales du développement du projet, cependant, elle permet de mieux comprendre et de mieux contrôler le développement du projet, tout en perfectionnant le processus de développement efficacement et qualitativement, avec des méthodes d'estimation qui se basent sur la connaissance préalable du développement du projet, et des données historiques des projets accomplis et l'expérience des estimateurs. Mais le développement du logiciel est une processus assez complexe, et il existe de

nombreux facteurs imprévisibles survenant pendant la construction logiciel; ceci résulte en une inexactitude d'estimation : de plus, l'essentiel de l'estimation est basé sur des affirmations parfois subjectives, ce qui est un autre facteur d'incertitude affectant la précision de l'estimation.

Dans le reste du mémoire, nous allons décrire, analyser et comparer les méthodes de mesure et les métriques employées dans la phase de construction du projet logiciel, en mettant l'accent sur les métriques appliquées pour mesurer la progression de la construction.

CHAPITRE II

REVUE ET ANALYSE DES MÉTRIQUES LOGICIELLES

Dans ce chapitre, nous présentons le processus de mesure du logiciel afin de montrer la conception, la définition et la classification des métriques logicielles, ensuite nous faisons la revue de toutes les métriques catégorisées en fonction de leur utilisation dans le domaine du génie logiciel. Nous offrons des analyses détaillées de ces divers types des métriques logicielles utilisées dans la construction du logiciel en soulignant celles qui sont impliquées dans la mesure de l'avancement de la construction.

2.1 Conception du processus de mesure en génie logiciel

Fenton a énoncé que la mesure logicielle a un rôle important pour le perfectionnement du processus de développement [30], parce que la mesure logicielle donne une visibilité sur le processus, le produit, les méthodes et les technologies employés dans le développement de logiciel. Elle nous aide à avoir des connaissances comme l'efficacité des outils de développement, la productivité de l'équipe de développement et la qualité du produit. De plus, la mesure logicielle nous permet d'établir une base de référence afin de comprendre la nature et les impacts des changements proposés par le gestionnaire du projet pendant le développement du projet. Finalement, elle permet aux gestionnaires de superviser et d'évaluer les effets des changements adoptés; les gestionnaires du projet pourront faire les ajustements appropriés le plus plutôt possible si les changements faits affectent négativement la qualité du processus de développement.

2.1.1 Le cadre d'établissement de la mesure logicielle

On présente un cadre de travail proposé par Fenton afin de décrire la création du processus de mesure et son application, il comporte trois principes exposés comme suit [30; 41]:

- Classer les entités en trois catégories: le processus logiciel[18], le produit logiciel[19] et les ressources logicielles[20], chaque entité est décrite par des attributs internes ou externes :

    - les attributs internes sont mesurés directement à partir d'une entité, par exemple la taille est un attribut interne d'un fichier de logiciel.

    - les attributs externes peuvent être mesurés indirectement en considérant la relation entre les entités et leur environnement d'application. Par exemple, la fiabilité[21] d'un projet tient compte non seulement du programme lui même, mais on doit aussi considérer des facteurs d'environnement comme le compilateur, la machine et l'utilisateur. La productivité est un autre attribut externe affecté par de nombreux facteurs, elle est mesurée indirectement par le rapport entre la taille du produit final et l'effort de développement.

Nous utilisons le tableau 2.1 qui suit pour décrire les entités et leur attributs dans le processus de mesure. Ce processus se fait dans toutes les phases de développement, et la plupart des mesures sont définies pour la phase de construction.

---

[18] Le processus logiciel est défini comme la collection des activités associées au développement du logiciel.

[19] Le produit logiciel comprend les artefacts ou documents produits par une activité dans le processus logiciel.

[20] Les ressources sont les entités exigées pour une activité du processus logiciel, par ex. : les ressources humaines, les matériels et les outils logiciels utilisés.

[21] La fiabilité fait référence à la capacité de fonctionnement d'un système informatique sans défaillance dans des conditions environnementales prédéfinies.

Entités	Attributs internes	Attributs externes
Produit		
Analyse des besoins	taille, réutilisation, fonction, modularité	Compréhensibilité, stabilité
Spécification	taille, réutilisation, fonction, modularité	Compréhensibilité, facilité de maintenance
Conception	taille, réutilisation, modularité, couplage, cohésion	Compréhensibilité, facilité de maintenance, qualité
Construction	taille, réutilisation, modularité, couplage, cohésion, complexité de flux de contrôle	Fiabilité, convivialité, réutilisabilité logiciel, facilité de maintenance
Test	taille, niveau de couverture du test	Qualité
Processus		
Analyse des besoins	délai et effort du développement	Efficacité en coût
Spécification	délai et effort du développement, nombre de changements d'exigence clientèle	Efficacité en coût
Conception	délai et effort du développement, nombre de changements d'exigence clientèle	Efficacité en coût
Test	délai et effort du développement, nombre de changements d'exigence clientèle	Efficacité en coût
Ressources		
Personnel	âge, coût	Productivité, expérience
L'équipe	taille, niveau de communication, structure d'équipe	Productivité
Logiciel	taille, prix	Convivialité, fiabilité
Matériel	prix, vitesse du CPU, taille de la mémoire	Convivialité, fiabilité

Tableau 2.1 Composants du processus de la mesure logicielle [30] (Fenton, 1997)

- Déterminer les objectifs de la mesure dans le développement du projet en utilisant l'approche GQM[22]. Cette dernière est un paradigme employant une liste contenant les

[22] GQM est l'abréviation de « Goal Question Metric »

objectifs majeurs possibles de la mesure à propos de l'amélioration de l'efficacité et de la productivité du processus de développement et la qualité requise pour le produit final. Un ensemble de questions seront posées en fonction des objectifs de mesure; ces questions touchent aux attributs exigés afin de décrire les objectifs de mesure, et on identifie les attributs mesurables en répondant adéquatement aux questions. Roche, Jackson et Shepperd [41] rapportent une étude comparative entre l'approche GQM et les autres alternatives de mesure appliquées dans l'industrie; ils ont constaté que la méthode GQM est la plus adaptable à l'amélioration du processus (ou du produit) logiciel, qu'elle offre une souplesse afin de définir et de valider les métriques logicielles pour les environnements particuliers, et qu'elle est facile à utiliser.

- Une étude de cas utilisant la méthode GQM dans la compagnie AT&T a été faite [30]; il s'agit de déterminer les attributs (ou les métriques) appropriés afin d'évaluer un processus d'inspection. Le tableau 2.2 ci-dessous décrit les objectifs de mesure, les questions posées et les attributs correspondant à chaque question.

Objectifs	**Questions posées**	**Attributs mesurables**
Planification du processus	Quel est le coût du processus?	Taux moyen d'effort par le nombre de lignes de code (Kilo Lines of Code --- KLOC), pourcentage de la réinspection
	Quel est le délai du processus d'inspection?	Taux moyen d'effort consommé par nombre de lignes de code (kloc), le nombre de lignes de code inspectées
Supervision et contrôle du processus d'inspection	Comment déterminer la qualité du logiciel inspecté?	Taux moyen des défauts détectés par le nombre de lignes de code (kloc), taux moyen d'inspection, taux moyen de préparation
	Comment superviser le statut du processus d'inspection?	Nombre de lignes de code inspectées dans le processus d'inspection
Perfectionnement du processus	Comment déterminer l'efficacité du processus?	Efficacité de correction des défauts, taux moyen des défauts détectés par le nombre de lignes de code (kloc), taux moyen du nombre de lignes de code inspecté,
	Quel est la productivité du processus?	Taux moyen d'effort consommé par chaque défaut détecté, taux moyen du nombre de lignes de code inspectées (kloc), taux moyen d'inspection, taux moyen de préparation

Tableau 2.2 Démonstration de l'exemple en utilisant l'approche GQM [30]

- Adapter le paradigme GQM en incorporant le processus CMMI[23]. Ce dernier propose un cadre permettant d'évaluer les niveaux de maturité au sein d'une entreprise sur une échelle de 1 à 5 définie dans le tableau ci- dessous. La mise en place du modèle CMMI représente une marche à suivre dans le but d'améliorer le processus de développement, en fournissant un ordre de capacité pour la réalisation des objectifs du développement. Selon Fenton [30], l'entreprise avec le niveau de maturité le plus élevé aura un processus de développement plus facile à prédire et plus contrôlable, et les attributs seront plus visibles aux développeurs durant le développement. À partir du tableau 2.3, les métriques adoptées passent des mesures d'estimation du niveau 1 aux mesures du produit du niveau 3; les mesures concernant le contrôle, la gestion et le perfectionnement de la qualité du processus de développement sont seulement considérées dans les niveaux 4 et 5. Brièvement, l'approche GQM nous aide à mieux comprendre et à déterminer les attributs des entités à mesurer; de plus, le processus CMMI nous recommande de rassembler et valider les métriques appropriées à chaque niveau de maturité afin de promouvoir la compréhension du processus de développement et le contrôle du projet.

Niveaux de maturité	Description
Niveau 1 Initial	Le processus n'est pas prévisible et contrôlable, les estimations sont faites afin d'établir une base de référence en termes de la taille du logiciel et de la charge du développement.
Niveau 2 Répétable	Les entrées et les sorties du processus, les contraintes et les ressources disponibles sont identifiables, les métriques de la gestion de projet sont utilisées.
Niveau 3 Défini	Les entrées, les sorties et les activités du processus de développement sont définies et documentées, les mesures du produit final sont considérables.
Niveau 4 Géré	Le processus est maîtrisé en appliquant la mesure logicielle : cependant, les déviations seront documentées et corrigées par l'adoption des changements dans le processus du projet, les métriques pour stabiliser et contrôler le processus de développement sont mises en place.
Niveau 5 En optimisation	Le processus est en amélioration de qualité, les ajustements sont adoptés continuellement afin de perfectionner le processus pendant le développement du projet, les données de mesure sont collectées pour indiquer quantitativement les faiblesses et les améliorations.

Tableau 2.3 Les cinq niveaux de maturité et leur description dans le modèle CMMI [41]

[23] CMMI est l'abréviation de « Capability Maturity Model Integration »

2.1.2 Une conception des métriques logicielles

Paul Goodman[40] a défini une métrique logicielle comme l'application continue des techniques de mesure, fournissant des informations de gestion significatives dans le temps, permettant d'améliorer le processus de développement et ainsi le produit logiciel. McConnell[42] a indiqué que le terme « métrique logicielle » comprend n'importe quelle mesure liée au processus de développement comme le nombre de lignes de code (KLOC), le nombre de défauts, le nombre de défauts par ligne de code, le nombre de défauts par module, le nombre de variables globales, les délais à compléter un projet, etc. tous des aspects mesurables d'un processus de développement. Au cours des dernières années, de plus en plus de grandes entreprises informatiques ont considéré la mesure logicielle comme une activité nécessaire au cours du développement du projet. Les métriques logicielles ont été adoptées largement dans l'industrie, mais il existe aussi des problématiques dans le domaine d'application des métriques logicielles [43].

Dans les applications des métriques en industrie, les entreprises n'importent pas de nouvelles métriques provenant de la recherche académique. En réalité, les métriques les plus adoptées en pratique sont déjà établies depuis une trentaine d'années. Le problème est que les métriques étudiées et définies par les académiciens ne sont pas jugées pertinentes pour la mesure logicielle dans l'industrie. Elles sont perçues comme ne servant pas les exigences industrielles, qui veulent des métriques pour perfectionner le processus de développement dans un contexte de gestion du développement. La plupart des activités de recherche se concentrent sur des métriques appliquées au code source détaillé du produit logiciel, qui possèdent moins d'utilité dans la pratique. De plus, les métriques fournies par la recherche académique sont fréquemment basées sur une petite portée d'application, et seront difficiles à réaliser et à calculer dans les gros systèmes logiciels de l'industrie.

Au dire de Paul Goodman [40], les applications des métriques logicielles se résument à quatre domaines cruciaux :

- *l'estimation du coût* du développement basé sur les modèles d'estimation comme le modèle COCOMO;

- *le contrôle du projet* utilisant des métriques afin de supporter quantitativement les activités de contrôle du projet, et qui incluent l'évaluation de la faisabilité du projet, la gestion du risque et la supervision de la progression du développement;

- *les métriques de la conception du projet* analysent la complexité du produit logiciel tout en mesurant la fiabilité, la facilité de maintenance et la compréhensibilité du système;

- *la provision d'information de la gestion du développement* comprenant la mesure de la productivité et de la qualité de l'équipe, l'efficacité du processus de développement, etc.

Goldense a interprété la conception de la métrique logicielle dans le développement simultané[24] du projet logiciel [44]:

- Les métriques au niveau de l'organisation doivent être considérées dans la mesure du développement logiciel. On estime la capacité de l'organisation par le ratio de dotation en personnel; cette métrique est calculée en faisant le rapport entre le nombre d'ingénieurs exigé et le nombre de fonctions du logiciel à implémenter. Ceci définit les ressources humaines exigées pour accomplir la tâche de développement.

- Les métriques sont employées pour établir les références de base du développement du projet en terme du coût et du temps de développement dans la planification du projet.

- Les métriques appelées «métriques de contrat» sont nécessaires dans le développement simultané. Elles sont déduites à partir des estimations validées par les équipes de développement autonomes de l'entreprise en fonction du délai à la livraison du produit au client, du coût des ressources matérielles et logicielles et de l'effort de développement. Après la négociation entre les équipes et le gestionnaire de l'entreprise, on détermine les métriques de contrat tout en quantifiant les intervalles pour les pourcentages des déviation calculés par: $(VO - VE)/VE$ où VO est la valeur obtenue durant le développement, et VE est la valeur estimée. Les équipes individuelles seront libres de rediriger le développement du projet sans supervision externe si les déviations des coûts (ou des délais) sont inférieures aux intervalles négociés; dans le cas contraire, le gestionnaire reprendra en charge le contrôle du développement.

---

[24] Dans le développement simultané, les personnes participent à la conception d'un projet logiciel afin d'assurer la cohérence de la construction et de la maintenance du projet logiciel, et travaillent ensuite parallèlement à la phase de construction du projet logiciel

- Les métriques mesurant la progression du développement du projet doivent être fournies, ces métriques sont dynamiques afin de suivre et de superviser la progression de la phase de construction du projet, elles servent aussi à calculer la date de fin du projet. La méthode du délai dynamique de livraison du produit au client est une mesure facile à calculer à l'aide des outils graphiques. Cette mesure suit la progression du développement du projet dynamiquement; on procède de la façon suivante :

    - Chaque fois que l'estimation de la date d'achèvement du projet est faite au cours du projet de construction, la date estimée sera notée sur le graphe. Au fur et à mesure de la progression de la construction, plusieurs estimations seront effectuées et notées; l'analyse de l'avancement du développement est faite à partir de ces dates estimées dans le graphe. Un exemple d'utilisation de cette méthode est illustré par la figure ci-dessous.

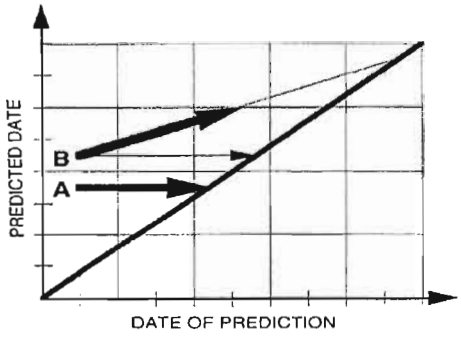

Figure 2.1 Exemple d'utilisation de la méthode du délai dynamique

Dans la figure 2.1, on observe que le développement du projet A se déroule bien, la tendance de développement se dirige horizontalement vers la ligne diagonale, ce qui signifie que le projet a bien respecté le délai planifié. En revanche, le projet B est en retard sur la planification, car le délai de livraison du développement du projet possèdent une tendance ascendante et asymptotique vers la ligne diagonale.

2.1.3 La classification des métriques logicielles

Afin de mener à bien le coeur de ce mémoire concernant les métriques appliquées dans la construction du logiciel, nous catégorisons ces diverses métriques logicielles en fonction de leurs objectifs de mesure ou de leurs utilisations. D'après Daskalantonakis [45], les métriques sont de trois types selon les utilisations des métrique logicielles. Ces trois types sont définis comme suit :

1. Métriques du processus logiciel --- ce genre de métrique logicielle est utilisé afin d'améliorer le processus de développement du projet comme la détection des défauts dans le développement et ainsi le coût d'un tel processus, etc.

2. Métriques du produit logiciel --- ce sont des métriques qui servent à améliorer le produit logiciel en terme de la compréhension du code, de la qualité du produit et de la facilité de maintenance. Elles comprennent des métriques comme la complexité de la conception du logiciel, la taille du code source et la convivialité de la documentation du produit.

3. Métriques du projet --- Les métriques du projet peuvent suivre et surveiller l'état du processus de développement en aidant les gestionnaires à prendre les dispositions adéquates au cours du développement du projet. Ce type de métrique comprend le nombre de développeurs, les efforts effectués dans chaque phase de développement et le ratio de réutilisation en conception du logiciel, etc.

Ces métriques offrent diverses utilisations dans le domaine de la mesure logicielle; elles sont réalisées à partir des programmes des métriques. Les objectifs de mesures de ces programmes déterminent les métriques adaptées afin d'effectuer l'étape de mesure; ils sont définis par l'approche GQM. Les consultants de la compagnie SPC [46] ont résumé les différents objectifs des métriques appliquées selon leurs expériences industrielles, ces objectifs et les métriques impliquées sont énumérés brièvement dans ce qui suit :

Amélioration du processus de développement --- cet objectif représente l'approche la plus prometteuse pour réduire les coûts de développement tout en maintenant la qualité du produit final. Les métriques représentent les mesures pour l'objectif comme le délai moyen écoulé pour l'identification et la correction des défauts, les efforts pour faire chaque activité du projet, le nombre de défauts détectés dans chaque activité du projet, le nombre des changements des exigences du client durant le développement, etc.

Amélioration de l'estimation logicielle --- comme nos constatations dans le chapitre précédent, les estimations précises en termes de la taille du produit, du coût de développement, du délai du développement sont des facteurs cruciaux pour réaliser un projet avec succès. Les métriques dans la démarche fournissent une référence de base comme les estimations et la taille réelle, l'effort en « homme-mois » réel du projet ou le délai réel du projet, le retard en « heures » de la planification du développement, le taux de travail représenté par « $^{Effort}/_{LOC}$ » pour chaque activité, la complexité du produit, etc.

Amélioration de la supervision du projet --- ces métriques sont dynamiques au cours du développement, leur valeurs changent avec la progression du projet tout en suivant l'état du développement du projet. Ces métriques sont représentées par le nombre de lignes de code écrites, le pourcentage de la complétion du travail pour chaque activité du projet, le pourcentage du budget dépensé par rapport au pourcentage planifié, le pourcentage du temps écoulé, etc.

Amélioration de la qualité du produit logiciel --- mesurer la qualité du logiciel est un processus très subjectif, la qualité est décrite par plusieurs attributs du produit logiciel comme la stabilité, la facilité de maintenance, l'extensibilité et la fiabilité. Il est difficile de trouver des métriques pour définir et mesurer ces attributs. Les indicateurs les plus adaptables pour la mesure de la qualité sont définis par le nombre de défauts détectés dans un projet, le temps moyen pour éliminer un défaut dans un projet, le nombre de défauts par ligne de code, le nombre de lignes de documentation du code et le pourcentage du code inspecté par le testeur logiciel.

Amélioration de la productivité --- la mesure de productivité est la quantité de travail réalisée par unité du temps. La stabilisation ou l'amélioration de la productivité est nécessaire pour mieux contrôler le processus de développement tout en réduisant la charge du développement du projet. Cette approche comprend des métriques comme le nombre de lignes de code produit par une personne en une heure (LOC/Homme*Heure), le pourcentage du budget disponible pour les outils de développement et ainsi le support du développement, la proportion des efforts dépensée pour la gestion du projet, etc.

2.1.4 Les métriques appliquées dans la construction du logiciel

Selon SWEBOK [3], le terme « construction de logiciel » concerne l'élaboration d'un produit logiciel qui comprend un ensemble d'activités comprenant totalement ou en partie l'implémentation, la vérification du logiciel, les tests unitaires, les tests d'intégration et le débogage. La phase de construction est étroitement liée à la conception de logiciel et au test du logiciel. Au cours du développement du logiciel, elle utilise le document de la conception logicielle comme ressource d'entrée principale et génère le produit logiciel pour la phase des

tests du logiciel; la définition du terme « construction de logiciel » et ses composantes dépendent du modèle de développement utilisé.

Dans les modèles linéaires de développement comme le modèle en cascade, on considère que la construction survient seulement après les activités préliminaires incluant l'analyse des besoins, la conception détaillée du logiciel et la planification du développement, le terme « construction de logiciel » est centré sur l'activité d'implémentation du code source. En revanche, dans les modèles itératifs comme le modèle de la programmation extrême, le modèle évolutionnaire, la construction de logiciel se présente simultanément avec les activités de développement préalables. Cependant, la construction est plutôt considérée comme une phase qui inclut des activités comme la conception du projet, l'implémentation et le test du logiciel. McConnell conclut que la construction de logiciel est l'activité centrale du processus de développement [42]. Selon la taille du projet, de 30 à 80% du temps de développement seront consacrés à la construction. Comme la construction affecte directement la qualité du produit final, le coût du développement du projet et la productivité des programmeurs, il est intéressant de perfectionner le processus de la construction de logiciel afin d'améliorer la qualité du logiciel, de réduire la charge du processus de développement et d'améliorer la productivité de l'équipe.

De nombreuses métriques ont été développées afin de mesurer le processus de construction et ainsi le produit achevé partiellement durant la construction [3]. Celles-ci comportent les entités mesurables comme le code source développé, le code source modifié, le code source réutilisé, le code source supprimé, la complexité du code source, les statistiques du code testé, le taux de correction des défauts, la charge du développement comprenant l'effort, le coût et la durée du développement. On classe les métriques affectées dans la construction selon leurs utilisations en deux catégories [47;48;50;51], comme ci-dessous :

- Pour le contrôle de la construction [47;48], une activité qui se répète lors du développement consiste à identifier les objectifs du produit logiciel, à établir la planification de la construction tout en estimant son coût et ainsi le temps nécessaire au développement, évaluer la performance du projet pour mesurer la progression de la construction. Les métriques adoptées servent d'indicateurs de performance du projet; elles sont appliquées et analysées en comparaison avec les valeurs planifiées au cours de la construction. Ensuite, on observe les variances significatives par rapport aux normes établies dans la planification; ces normes sont les durées, les

budgets disponibles et les objectifs qualitatifs du produit logiciel. Après avoir fait l'évaluation du processus de construction, des ajustements dans la planification du projet seront proposés. Ebert a illustré le contrôle de la construction par la figure 2.2 ci-dessous.

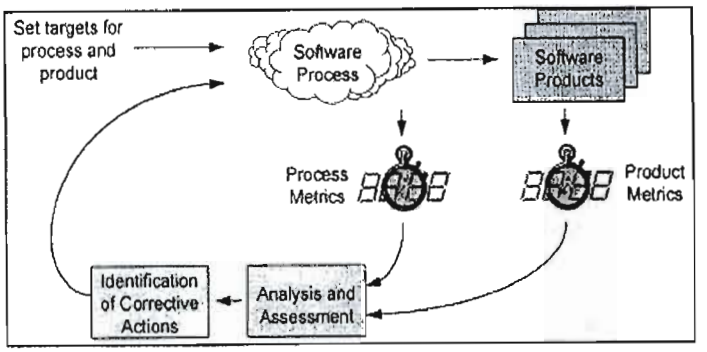

Figure 2.2 La gestion du projet conduite par les métriques selon Christof Ebert [48]

À partir de la figure 2.2, on voit que le contrôle de la construction s'adapte répétitivement tout au cours du processus de développement. En effet, il assure que le projet va dans la bonne direction, que les objectifs de performance et les budgets commerciaux définis dans la planification sont respectés. Ceci procure un support significatif afin d'améliorer continuellement le processus du projet. Cependant, les métriques employées dans le contrôle de la construction mesurent la progression de la construction, ce qui permet d'avoir une bonne connaissance du projet de construction et de rendre le processus de construction plus compréhensible. Les mesures sont considérées comme des évidences numériques des activités d'amélioration du processus logiciel comme la prise en décision, l'évaluation de la performance, la planification du projet, les dispositions préventives afin de minimiser les risques possibles dans la future construction, etc.

- Pour l'assurance de la qualité du produit logiciel au cours de la construction, selon la norme internationale ISO9126[25] définit par IEEE [30], le terme « qualité logicielle » comprend les six caractéristiques d'un produit logiciel final offrant des fonctions se conformant aux exigences des clients:
  - la capacité fonctionnelle du logiciel

[25] La norme ISO9126 comprend le modèle de qualité logicielle; ce dernier initialement réalisé par McCall en 1977 a été adopté comme une norme internationale par IEEE en 1991.

- la fiabilité du logiciel [26]
- l'efficacité du logiciel
- la convivialité du logiciel
- la facilité de la maintenance du produit livrable
- la portabilité du logiciel[27]

Ces attributs de la qualité ci-dessus décrivent le produit logiciel livrable dans la phase de maintenance; Kan [50] a mentionné que la qualité du logiciel doit être prise en compte durant tout le cycle de vie du logiciel afin d'améliorer la qualité du produit final et augmenter la satisfaction du client. Ceci est dû au fait que les interrelations des attributs ne sont pas toujours proportionnelles durant le développement; par exemple, le produit logiciel ayant plus de complexité fonctionnelle dans la conception possèdera plus de capacité, mais ceci provoquera également plus de difficulté pour la maintenance du système. En fonction du type d'application et du type des clients, nous devons également définir les différents objectifs pour ces divers attributs de qualité afin de répondre aux exigences du client dans la planification du développement.

L'autre vue proposée par Kan est que la qualité logicielle doit comprendre non seulement les attributs de qualité du produit final, mais encore la qualité du processus de construction. En effet, la construction est un processus complexe, qui comprend une série d'activités, chaque activité produisant un objet intermédiaire possédant un certain niveau de qualité ce qui affecte la qualité du produit final. Cette dernière est influencée par de nombreux facteurs négatifs, il est donc significatif de mesurer le processus du développement et ainsi les attributs de la qualité pendant chaque activité. La mesure de qualité sert à l'assurance de qualité afin de se conformer aux objectifs qualitatifs du produit final durant la construction. La mesure de qualité logicielle est étroitement associée à deux types de métriques principalement:

1. Les métriques de qualité du produit final qui consistent à mesurer la qualité inhérente du produit par le nombre de défauts détectés dans le logiciel, le temps moyen de panne du logiciel, etc. Également ce type de métriques décrit et mesure la satisfaction du client par les problèmes rapportés par les clients et le pourcentage de satisfaction des clients, etc.

---

[26] La fiabilité reflète le fait que le système du logiciel maintient son niveau de performance sous certain circonstance pour une période du temps.

[27] La portabilité du logiciel veut dire la capacité d'un système d'être transféré dans différents environnements d'application

2. Les métriques dans le processus de construction : ce type de métrique de qualité vise à offrir une meilleure compréhension du processus de construction afin d'améliorer la qualité du processus. Des variantes des métriques ont étés établies et validées dans cette démarche, elles sont basées sur le fonctionnement externe des objets intermédiaires durant la construction selon la densité de défauts détectés durant le test formel du système, les défauts corrigés dans chaque phase de développement et l'efficacité de la correction de défaut, etc. D'autre part, on devra aussi considérer les caractéristiques procédurales du projet affectant directement la qualité du produit final; ceci réfère aux ressources comme le nombre de développeurs et leur expérience, la planification et la taille du projet, la structure des équipes de développement, etc.

   Par ailleurs, la mesure de la complexité tend à fournir une vue interne sur la qualité logicielle dans la conception et l'implémentation du produit. Cependant, la mesure est basée sur des modules du projet; elle permet d'analyser et quantifier la structure, la difficulté et la facilité de maintenance des composants du projet par les métriques comme la complexité cyclomatique, la métrique scientifique (Halstead), la métrique de la structure, etc. Ces métriques procurent des indices qualitatifs aux développeurs afin d'améliorer la qualité du produit final en réduisant la complexité.

Dans la section suivante, nous allons présenter et analyser un ensemble de métriques ainsi que leurs applications réalisées dans la construction de logiciel pour mesurer les entités logicielles du type processus, objet logiciel ou ressource disponible durant la construction, et ce afin de prendre les décisions convenables pour les améliorer en terme de la qualité du produit final et de la performance du processus. Cependant, la même métrique peut servir à plusieurs objectifs de mesure, on effectue l'évaluation de chaque métrique en s'appuyant sur la convivialité de leur propres adaptions industrielles. Nous analysons comparativement la performance des métriques pour le même objectif de mesure, l'analyse se basant sur certaines études de cas empiriques. Comme les informations sur la progression de la construction seront exigées et recueillies durant la construction, elles jouent un rôle effectif et positif pour la prise de décision dans le contrôle du développement. Nous soulignons la présentation des métriques et les alternatives adaptables dans le but de mesurer l'avancement de la construction tout en obtenant des données précieuses qui aident le contrôle du projet quantitativement et qualitativement. Nous allons également évaluer ces métriques ou méthodes dans le contexte de la gestion du projet.

2.2 Revue des métriques logicielles

En partant du tableau 2.1 établi par Fenton dans la section 2.1.1, nous commençons la revue à partir des métriques ayant des attributs stables (externes ou internes) du produit ou objet logiciel. Par la suite, nous introduisons des métriques ou des alternatives tout en quantifiant les caractéristiques du processus de construction, et nous analysons des métriques identifiant les ressources du processus. Cependant, nous mettons l'accent sur les métriques qui peuvent être adaptées pour mesurer la progression de la construction.

2.2.1 Métriques pour mesurer la dimension logicielle

Selon Fenton [30], la dimension est un attribut interne du produit logiciel, qui peut être mesurée statiquement sans exécution du système. La dimension d'un produit logiciel est identifiée par trois aspects spécifiques: la taille physique du système, les fonctions développées dans le système et enfin la complexité du système logiciel. Il existe de nombreuses métriques et méthodes pour calculer la dimension logicielle, on les regroupe ci-dessous en trois sections principales [30; 31]:

- La méthode des points de fonction permet de mesurer les fonctions à partir de la spécification formelle du système; la méthode est également applicable à la conception et au code du système
- Les métriques pour mesurer la taille du code du système au cours de la construction logicielle
- La mesure de la complexité du système peut être interprétée par plusieurs aspects :
    - La mesure de la complexité du problème à résoudre;
    - La complexité de l'algorithme réalisé par le code, qui se réfère à l'efficacité de l'objet logiciel;
    - La complexité de la structure du système à réaliser. Les métriques classiques de ce type sont la mesure des structures de contrôle de flux, la structure hiérarchique du système et le couplage entre les modules du système, etc.

2.2.2 Métriques de taille de l'objet logiciel

Les métriques de taille sont utilisées pour obtenir la taille des produits logiciels générés dans les phases de spécification, de conception et de construction. Les mesures établies dans les phases préliminaires permettent de prédire l'effort exigé pour la production du logiciel. Nous avons parlé de ces métriques et méthodes dans la section 1.4.3 du chapitre I. Pour la construction, diverses métriques et alternatives sont développées afin d'indiquer la taille du code produit dans la réalisation du logiciel; on les résume dans ce qui suit [30;31].

Nombre de lignes de code

C'est la métrique qui est employée le plus souvent dans l'industrie pour la taille des gros systèmes informatiques, et qui est souvent repérée par l'abréviation LOC « *Lines Of Code* », ou en unité de milliers des lignes de code KLOC « *Kilo Lines Of Code* ». En fonction des multiples utilisations de la métrique, plusieurs règles différentes sur le comptage du nombre de lignes de code ont étés définies par des chercheurs académiques et des organisations informatiques [30;31].

- En 1986, Conte, Dunsome et Shen [31] ont affirmé essentiellement qu'une ligne du code est toute ligne du texte du programme à l'exception des lignes du type commentaire ou de ligne vide. Cela veut dire que le compte du nombre de lignes de code inclura tous les lignes contenant des en-têtes, des déclarations et des instructions exécutables ou non exécutables du code produit mais sans les lignes commentaires et les lignes vides. En 1987, Grady et Caswell [30] de la compagnie Hewlett-Packard ont défini la ligne de code comme une ligne d'instruction qui n'est pas un commentaire ou une ligne vide dans le programme; elle est repérée par l'abréviation NCLOC (*Non Comment Line Of Code*). Cette définition est couramment acceptée et utilisée dans l'industrie informatique.

- Actuellement[30], les chercheurs et les gens de l'industrie tendent à définir la métrique comme le nombre de lignes de code livrables, c'est à dire le compte de lignes de code considère uniquement le nombre de lignes d'instructions exécutables

en ignorant les lignes de commentaire, les déclarations et les en-têtes dans le code source.

Analyse de la métrique

Nous procédons à l'analyse par le constat de Conte, Dunsmore et Shen[31] qui affirment que la mesure du nombre de lignes de code n'est pas cohérent parce qu'il y a des lignes du code qui sont plus difficiles à produire que d'autres. Fenton [30] note que la définition de la ligne de code est évidemment influencée par son utilisation : en effet, le nombre des lignes de code non commentaires (NCLOC) défini par Hewlett-Packard est né du besoin d'évaluer la productivité. Cette dernière est calculée sous la forme « *NCLOC / Homme * Mois* », et est une métrique valide pour la mesure de la productivité, car elle est liée aux efforts des développeurs. D'autre part, on devra tenir compte du nombre de lignes de commentaires si on utilise la métrique LOC pour identifier la densité des commentaires qui est une attribut lié à la facilité de la maintenance du logiciel, et qui est mesurée par *CLOC / LOC* où CLOC signifie le nombre de lignes de commentaires.

Dans la pratique industrielle, le nombre d'instructions exécutables est une métrique importante qui permet de mesurer la densité des défauts. Celle-ci est calculée par *Defect / ES* où le terme « Defect » représente le nombre de défauts détectés et le « ES » (*Executable Statement*) est le nombre des instructions exécutables n'incluant pas les lignes de commentaires, les déclarations des données et les en-têtes du code source produit. En revanche, dans la pratique les développeurs veulent souvent distinguer les lignes de code livrées aux clients de la quantité totale du code source développé. En visant cet objectif d'utilisation, la métrique LOC doit comprendre les déclarations et les en-têtes, ainsi que les instructions exécutables du code source. À partir de la revue de [30;31;51], nous pouvons faire quelques remarques sur cette métrique dans ce qui suit :

- Le nombre de ligne de code montre une forte variabilité selon les divers langages de programmation utilisés [31]. Cependant, certains langages permettent d'avoir deux ou plusieurs instructions sur une seule ligne du code ou une seule instruction peut aussi occuper deux ou plusieurs lignes. Ceci provoque une incohérence évidente du nombre de lignes de code dans ces divers langages de programmation.

- Selon le résumé de Jarrett [51] sur la définition des lignes de code, on note qu'il y a peu de différence entre les nombres de lignes de code obtenues par différentes manières de compter. Une étude de cas a été réalisée par Rosenberg afin de prouver l'observation; la figure ci-dessous donne les résultats empiriques de l'étude de cas.

Application	# of Files	Median NCSL: As-is	Median NCSL: Max	Median NCSL: Min	Median NCSL: cpp
Axe	86	38	44	38	50
Chipmunk	167	108	126	108	708
DGL	125	51	51	46	215
Gnuplot	46	324	344	316	319
Moira	198	100	112	100	184
Netrek-client	83	202	218	199	1077
Netrek-server	95	151	168	147	683
Oleo	109	153	153	134	205
Postgres	753	76	81	76	246
Spice	891	59	63	60	568
Tcl	47	167	205	167	633
Tex	50	1640	1652	1463	2425

Figure 2.3 Statistiques des moyennes de NCLOC de chaque application selon Rosenberg, Jarret [51]

- Pour expliquer les données de la figure 2.3, cette étude expérimentale s'est déroulée en utilisant 12 différents types d'application écrites en langage C. La colonne « As-Is » est la moyenne du nombre de lignes de code sans commentaire parmi les 86 fichiers sources dans l'application « Axe », le paramètre « As-Is » représente le format du code source utilisant le style original; le paramètre « Max » représente un format du code d'un style plus formel et plus lisible qui inclut le nombre de lignes vides et les lignes de commentaires maximum; le paramètre « Min » représente le code source avec un style moins formel et aussi moins compréhensible ayant le nombre minimum de lignes de code; le « Cpp » représente l'utilisation partielle du pré processeur dans les fichiers sources; le prétraitement consiste à générer et inclure dans le fichier source des directives comme les fichiers d'en-têtes, et les expansions de macros. À partir du tableau de la figure ci-dessus, Rosenberg trouve que les différences des nombres de lignes de code selon les différents styles tels que « As-Is » ou « Max » ou « Min » ne sont pas significatives. En conséquence, on pourra constater que le style de formation des lignes de code n'affecte essentiellement pas le nombre de lignes de code. Également, il remarque que le processus de prétraitement augmente significativement le nombre de lignes de code, et que la structure du programme après le prétraitement doit être prise en compte dans la mesure, pour compenser son effet négatif sur la technique de comptage des lignes de code.

- Le nombre de lignes de code permet diverses utilisations pour la mesure logicielle, qui sont divisées par Jarrett [51] en trois démarches principales :

- Dans l'estimation de l'effort de développement et maintenance, on s'en sert comme variable indépendante dans les modèles de régression d'estimation comme les modèles COCOMO, etc.

- Il est souvent considéré comme une covariable pour les autres métriques. Un ensemble de métriques sont corrélées avec la métrique LOC comme la productivité, la densité des défauts et la densité des commentaires, etc. Rosenberg conclut que la relation entre la densité des défauts représenté par *Defect / LOC* et le nombre de lignes de code, est une corrélation négative, c'est–à-dire que le programme avec une plus grande quantité de code source possède une densité du code moindre. Ceci est prouvé par des simulations mathématiques sur un ensemble d'échantillons de données d'entrée avec un nombre de défauts et un LOC générés au hasard.

- Dernièrement, la métrique du nombre de lignes de code est également utilisable comme standard en établissant des normes qui sont comparables aux autres métriques, afin de les évaluer.

Mesures issues de la science logicielle d'Halstead

Ce modèle a été développé par Maurice Halstead en 1976, et comprend un ensemble de métriques pour mesurer la longueur du logiciel, le volume du système logiciel et la complexité du logiciel pour prédire le temps et l'effort de développement [30;31]. Un programme est défini comme une collection d'unités lexicales divisées en opérateurs et en opérandes. On identifie ces derniers par les symboles suivants [31]:

μ_1 = le nombre d'opérateurs uniques

μ_2 = le nombre d'opérandes uniques

N_1 = le nombre total d'apparitions des opérateurs

N_2 = le nombre total d'apparitions des opérandes

La longueur du logiciel est identifiée par le nombre $N = N_1 + N_2$; le vocabulaire μ est défini par $\mu = \mu_1 + \mu_2$; le volume V du logiciel représente le nombre de bits nécessaires pour stocker le programme entier dans la mémoire. Il est calculé de la façon suivante :

$V = N \times \log_2 \mu$ où chaque élément (opérateur ou opérande) du vocabulaire μ réserve $\log_2 \mu$ bits, et $N \times \log_2 \mu$ est le nombre total de bits exigés pour le programme de longueur N.

Également dans ce modèle, Halstead a établi des métriques pour estimer la longueur du système, quantifier et déduire le niveau du programme afin d'estimer l'effort et le temps nécessaires à la construction.

Tout d'abord, Halstead définit la formule prédisant la longueur du programme :

$$N' = \mu_1 \times \log_2 \mu_1 + \mu_2 \times \log_2 \mu_2$$

Par la suite, le niveau du programme L est calculé par la formule suivante:

$L = V^* / V$ où V^* indique le volume potentiel du programme, qui est la longueur minimale du code exigée pour implémenter le programme, et V est le volume implémenté du programme. Le niveau L est le rapport proportionnel entre ces deux types de volume. La difficulté D du programme est définie par l'inverse du niveau du programme comme $D = 1/L$. Au dire de Conte, Dunsome et Shen [31], lors de l'augmentation du volume V dans le programme, le niveau L diminue et la difficulté croît. Il est souvent très difficile d'identifier le volume potentiel du programme; Halstead offre aussi une formule mathématique pour déduire le niveau du programme:

$L' = 1/D = \frac{2}{\mu_1} \times \frac{\mu_2}{N_2}$ où $\frac{\mu_1}{2}$ est le niveau relatif de difficulté dû aux opérateurs additionnels ajoutés dans le programme, et $\frac{N_2}{\mu_2}$ est le nombre moyen de fois où un opérande est utilisé.

En fonction du niveau présumé, on pourra estimer l'effort de construction. Halstead considère l'effort de construction lié à deux attributs : le volume V qui est corrélé

proportionnellement avec l'effort et le niveau du programme qui a une relation négative avec l'effort à dépenser. L'estimation de l'effort est illustrée par la formule suivante :

$E = V/L' = \frac{\mu_1 N_2 N \log_2 \mu}{2\mu_2}$ où l'unité de mesure d'effort est le EMD[28] qui introduit les efforts nécessaires afin d'achever le programme.

Finalement, le temps prévu T pour l'achèvement du programme sera donné par la forme suivante :

$T = E/\beta$ tels que le cœfficient β nommé le nombre de Stroud représente les EMDs consommés du programmeur par seconde, la valeur du β varie dans l'intervalle[5, 20], la valeur la plus applicable étant en général 18.

Analyse du modèle de Halstead

À partir de la discussion de Fenton [30] et de celle de Kan [49] , on apporte les conclusions analytiques suivantes :

- Le modèle de la science logicielle comprend une série de métriques basées sur des modèles mathématiques; il manque une évidence empirique d'un système informatique réel afin de valider ce modèle; de plus, les équations des procédures d'estimation de la longueur, de l'effort et du temps sont modélisées de façon compliquée et offrent une faible utilité de mesure durant la construction. Ceci est dû au fait que ces prédictions exigent la disponibilité des vocabulaires μ_1 et μ_2, des longueurs N_1 et N_2, du volume V ; ces derniers sont déterminés lors de l'achèvement ou bien à un moment proche de la complétion de la construction, ce qui rétrécit la portée d'utilisation des équations prédictives.
- Le modèle de science logicielle a un impact significatif dans la mesure de logiciel; elle a été une approche controversée et disputée par les informaticiens académiques pendant des années. Cependant, du côté positif, les métriques en unités lexicales

[28] EMD est l'abréviation de « Elementary Mental Discrimination », il présente l'unité de mesure en effort exigible afin de comprendre le programme.

comme la longueur N , le vocabulaire μ et le volume V sont des mesures raisonnables et acceptables pour quantifier les attributs internes de la dimension du logiciel.

2.2.3 La mesure de la taille fonctionnelle logicielle

La mesure des points de fonction est la méthode principale pour mesurer la fonction inhérente du produit logiciel. Elle est faite à partir des documents de spécification produits dans les phases préliminaires du développement [30]. Durant la construction du logiciel, la métrique du nombre de points de fonction identifie et exprime la taille fonctionnelle du système. Elle est citée dans ce qui suit [23,30,52].

Premièrement, pendant l'implémentation du système logiciel, le nombre de points de fonction peut être utilisé afin d'exprimer la productivité qui est représentée par le nombre de points de fonction achevé par heure de travail. Également, cette métrique est souvent utilisée pour calculer la densité des défauts en divisant le nombre de défauts par le nombre de points de fonction. D'un autre coté, on peut suivre la progression ou la complétion de la construction par examen du nombre de points de fonction spécifiés, conçus, implémentés et testés, qui sont identifiés dans les différentes activités de la construction.

Deuxièmement, des points de fonction peuvent être obtenus dans les phases initiales du développement comme la spécification ou la conception du système. Ils seront utilisés comme paramètres clefs des activités d'estimation de l'effort ou du coût de développement. Les chercheurs universitaires observent que le passage du nombre de points de fonction à l'effort de développement peut se faire par un modèle de régression linéaire, une régression exponentielle ou bien par un ratio dépendant du type du projet, etc.

La méthode des points de fonction a été initialement formulée par Albrecht en 1979 (nous l'avons présentée dans la section 1.4.3.3 du chapitre précédent). Cependant, la spécification du système à implémenter sera découpée en plusieurs composants selon des types prédéfinis. Les estimateurs associent une complexité pondérée à chaque type de composant subjectivement; après avoir compté et calculé le nombre points de fonction de chaque type, on obtient le nombre de points de fonction brut. Ce dernier sera évalué et calibré dans le processus d'ajustement; au cours des dernières années, l'approche d'Albrecht a été reprise et

a évolué en d'autres méthodes de points de fonction en vue de mesurer la taille fonctionnelle du système plus adéquatement et plus précisément. Cependant, différentes propositions ont été mises en place afin d'améliorer le processus d'ajustement [52].

Le processus d'ajustement dans la mesure des points de fonction

L'essentiel du processus est d'appliquer la collection des variables GSC « General System Characteristics » pour calibrer la taille fonctionnelle en points de fonction bruts. Les variables GSC tentent d'évaluer la complexité du produit logiciel et aussi la complexité de l'environnement d'application. Dans l'approche d'Albrecht, version 1984 [26;30], les facteurs de complexité GSC sont fondés sur 14 caractéristiques du système. Ils comprennent des sujets pertinents comme recouvrement du système, communication de données, fonctions distribuées, exigences de performance, configuration largement utilisée, entrée de données en-ligne, facilité d'opération, mise à jour en-ligne, traitement complexe, réutilisation du logiciel, facilité d'installation, sites multiples et modifications faciles. Chaque facteur des GSC est évalué et associé à un degré d'influence qui possède une variation de 0 à 5; la valeur d'ajustement VAF du processus est calculée par la formule suivante :

$VAF = 0.65 + 0.01 * \sum C_i$ où $\sum C_i$ est la somme des degrés des variables GSC, la valeur 0.01 représente le même poids associé à chaque facteur des GSC, enfin, la valeur VAF sera multipliée par le nombre de points de fonction brut pour obtenir le nombre de points de fonction ajusté.

Analyse de la mesure des point de fonction dans l'estimation logicielle

Malgré les facteurs subjectifs de la mesure introduits par les jugements subjectifs des différents estimateurs dans le comptage du nombre brut des points de fonction et l'évaluation des facteurs de complexité des GSC, le bénéfice majeur de la méthode des points de fonction réside dans le fait qu'elle prévoit et produit la taille logicielle estimée en nombre de points de fonction dans les phases initiales du développement d'une manière plus rigoureuse que le nombre de lignes de code. Nous présentons le tableau 2.4 ci-dessous qui montre les

avantages de la métrique des points de fonction par rapport à la métrique du nombre de lignes de code dans l'estimation logicielle[23]:

La méthode des points de fonction	Le nombre de lignes de code
• Indépendante des langages, des outils ou des méthodologies de programmation utilisés • Estime la taille fonctionnelle à partir des fichiers de spécification des besoins du client dans les phases initiales de développement, facile à adapter au changement des besoins et ainsi de refaire la prédiction de la taille • Estimation basée sur la vue externe de l'utilisateur représentant les fonctions du système; c'est une mesure plus cohérente et plus facile à comprendre par des estimateurs et des organisations différents	• Forte variabilité sur le nombre de lignes de code des projets construits pour différents langages de programmation; il est impossible de comparer la taille en lignes de code sur des projets développés en différents langages • Manque d'une définition uniforme sur le sens d'une ligne de code et il existe de nombreuses règles de comptage de lignes de code. Il est donc très difficile de faire des études comparatives en utilisant le nombre de lignes de code comme une mesure de la taille logicielle • Très difficile de prévoir la taille en nombre de lignes de code précisément en fonction des spécifications ou de la conception du système à implémenter

Tableau 2.4 Comparaison entre la méthode des points de fonction et celle du nombre de lignes de code

Dans l'estimation de l'effort de développement, plusieurs modèles de régression empiriques ont été réalisés afin de prédire l'effort en prenant la taille logicielle en nombre de points de fonction comme variable indépendante. Les données supportant ces modèles sont obtenues à partir d'ensembles d'échantillons limités [23]; donc l'utilité de ces modèles est restreinte à certains contextes tels l'environnement de l'application et le type du système logiciel. Cependant, on cite les modèles d'estimation suivants [23]:

- Albrecht et Gaffney produisent le modèle de régression linéaire basé sur 24 applications développées par IBM en 1983. Il est formulé comme suit : $E = -13.39 + 0.0545 \times PF$ où PF est le nombre de points de fonction, la variable dépendante E représente l'effort estimé.

- Kemerer a développé un autre modèle d'estimation en fonction de données provenant de 14 projets d'une compagnie. Dans ce modèle, l'effort estimé est calculé par

l'équation polynomiale: $E = 60.62 + 7.728 \times PF^3$; les résultats expérimentaux montrent que ce modèle estime l'effort plus précisément que celui d'Albrecht et Gaffney.

- Matson, Barrette et Mellichamp suggèrent un modèle de régression linéaire plus fidèle et plus performant que les deux modèles présentés ci-dessus. L'établissement du modèle se base sur un échantillon de donnée de 104 projets qui présentent une plus grande échelle; le modèle est exprimé par l'équation suivante : $E = 585.7 + 15.12 \times PF$

Analyse de la méthode des points de fonction

D'après les revues [23;30;52], des études analytiques ont été faites par des chercheurs universitaires et des gens de l'industrie sur l'approche originale d'Albrecht dans une perspective de mesure. Elles ont identifié de multiples contraintes et limitations qui existent dans l'approche d'Albrecht; pourtant, de nouvelles techniques de points de fonction ont été suggérées par eux en apportant des modifications à l'approche originale. Ces dernières affectent surtout le processus d'ajustement de la méthode, plus particulièrement la définition des variables GSC impliquées et ainsi l'évaluation de degré d'influence de chaque GSC afin de produire la value d'ajustement VAF plus adéquatement. Dans ce qui suit, on résume les discussions sur les limitations potentielles de la méthode originale:

- Dans la méthode des points de fonction « MarkII » définie par Symons[30;52], on trouve que 14 variables GSC ne sont pas suffisantes pour décrire la complexité de l'environnement. Il ajoute encore 5 caractéristiques du système comme l'interaction avec d'autres systèmes, la sécurité, la documentation, etc. Il indique également qu'il n'est pas adéquat de pondérer du même poids 0,01 chaque caractéristique GSC. Au contraire, dans l'étude de cas faite par Kitchenham et Lokan [52], ils observent qu'il y a seulement 5 ou 6 caractéristiques GSC impliquées dans le processus d'ajustement. Benyhaia et al. [52] trouvent qu'il est difficile de différentier ces 14 variables GSC prédéfinies, uniquement 7 des caractéristiques sont significatives dans leurs échantillons de données.

- Kitchenham, Pfleeger et Fenton [30;52] ont fait une étude analytique pour évaluer le processus d'ajustement représenté par la formule $VAF = 0.65 + 0.01 * \sum C_i$. L'analyse est basée sur une perspective de la théorie de mesure, et ils constatent que le calcul de la valeur d'ajustement VAF se fait d'une manière incohérente en combinant les mesures de différentes échelles. Cela veut dire que le degré C_i de chaque caractéristique GSC est classé dans l'échelle ordinale avec un ordre croissant de 0 à 5. Il est indécent de transformer ces nombres de l'échelle ordinale en valeurs

numériques de l'échelle rapport, parce qu'ils ne sont pas du même type d'échelle et représentent différentes magnitudes. De plus, il est inadéquat et sans signification d'interpréter les 14 caractéristiques avec le même poids 0,01, parce que chaque facteur de complexité GSC a différents impacts sur la taille fonctionnelle du système final.

- Au dire de Kitchenham[30;52], les caractéristiques GSC comme les exigences de performance et la complexité de traitement sont des facteurs de complexité pris en compte dans le processus d'ajustement. Ces facteurs seront considérés encore une fois dans les modèles d'estimation de l'effort en paramétrant le nombre de points de fonction ajusté comme variable indépendante; on risque donc de doubler l'ajustement en prenant ces facteurs de complexité une deuxième fois, ce qui réduit la précision de l'estimation de l'effort.

2.2.4 La mesure de la complexité logicielle

Selon le schème de classification déjà cité dans la section 2.2.1, la complexité logicielle peut être divisée en trois catégories : la complexité structurelle, la complexité algorithmique et la complexité du problème à résoudre. Dans cette section, nos intérêts portent plus particulièrement sur les métriques de complexité structurelle qui tentent de décrire les attributs internes dans la conception ou le code source du système. L'objectif de l'approche est d'établir une relation entre la structure du produit logiciel et la qualité du logiciel comme la fiabilité, la testabilité et la facilité de maintenance du système. Les mesures de complexité structurelle sont dérivées de la spécification des exigences, de la conception et du code; elles nous permettent de comprendre et d'identifier les facteurs négatifs qui entraîneront plus de défauts du logiciel ou moins de productivité durant la construction. Nous classifions et présentons les métriques structurelles dans ce qui suit [30;49;53;54;55]:

- La mesure de la structure de contrôle du flux : les métriques de ce type sont employées afin de déterminer la complexité de la structure de contrôle à l'intérieur de chaque module[29] individuel du système; la complexité de McCabe est la métrique principale dans cette démarche.

- La mesure de la structure de flux de données : ce type de mesure est censé identifier et quantifier les attributs des interrelations entre les modules du système en capturant

---

[29] Le terme « module » est défini comme un ensemble d'instructions exécutables qui possède un paramètre d'entrée et un résultat en sortie [53].

les flux d'information entrée/sortie d'un module. Ce genre de mesure comprend les métriques comme la métrique de flux d'information d'un module individuel et la complexité globale du système.

- Les métriques dans la conception orientée objet : ces dernières tentent d'élaborer le système avec des objets qui encapsulent des états et des caractéristiques et des opérations. Cela est différent de l'approche de conception fonctionnelle, qui décompose hiérarchiquement le système en modules et en procédures qui possèdent les différentes fonctions du système. Afin d'évaluer la complexité de la structure de conception orientée objet, un ensemble de métriques spécifiques sont réalisées pour mesurer les concepts et les caractéristiques orientées objets qui comprennent méthodes, classes, héritage et couplage entre des classes.

2.2.4.1 La mesure de complexité de McCabe

La métrique de la complexité cyclomatique a été introduite par McCabe en 1976 [49;53;54]; cette mesure évalue les structures de contrôle d'un module représenté par un graphe connexe G illustré par la figure 2.4. Dans le graphe, chaque noeud N représente un module fonctionnel du programme, les noeuds T_i représentent des blocs d'instructions contigus; les arêtes du graphe sont spécifiées par les prédicats des branchements conditionnels incluant les structures comme «si-alors-sinon», boucle «tant que» et boucle « répète...jusqu'à», etc. On détermine la complexité structurelle en calculant le nombre cyclomatique V ; ce dernier terme peut être exprimé comme le nombre maximum de circuits qui sont indépendants linéairement. Il peut être calculé de deux différentes manières présentées ci-dessous.

$V = e - n + 2$ ou $V = P + 1$ où e est le nombre des arêtes disponibles dans le graphe, n est le nombre des noeuds du graphe, enfin P indique le nombre des prédicats des branchements conditionnels.

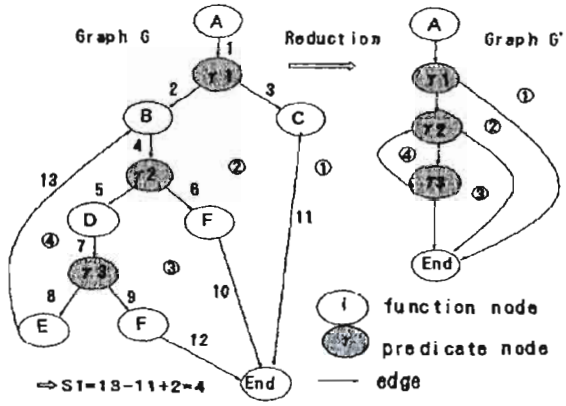

Figure 2.4 Exemple d'un graphe de contrôle de flux adapté de Takahashi [54]

Une extension de la mesure est présentée dans le graphe G' dans lequel la présentation graphique de la complexité est simplifiée en éliminant les noeuds des modules intermédiaires qui sont redirigés ou définis par les branchements conditionnels, car on considère seulement les noeuds des prédicats qui sont les seuls paramètres influençant la complexité structurelle du programme.

Analyse de la mesure de complexité cyclomatique de McCabe

La mesure de la complexité cyclomatique fournit une indication prédictive des attributs qualitatifs du système comme la testabilité et la facilité de maintenance du module; elle permet aux testeurs de logiciel d'identifier les modules plus complexes et difficiles qui exigent plus d'effort pour les inspecter et les comprendre; elle peut être également utilisée pour prédire l'effort d'implantation ou de test du système. Récemment, des études empiriques ont été faites afin d'explorer la corrélation entre la complexité de McCabe et la qualité du système représentée par le nombre de défauts ou la densité de défauts du programme, nombre de défauts par millier de lignes de code (KLOC). En résumé, nous citons les observations expérimentales suivantes [49; 54; 56; 57] :

- Les observations sur la corrélation entre la complexité cyclomatique et le nombre de défauts ou la densité de défauts des programmes ne sont pas consistantes :
    - Selon les analyses statistiques faites par Khoshgoftaar et Munson [56], la taille du module en nombre de lignes de code est fortement liée à la

complexité cyclomatique de manière linéaire. Elle est plus représentative pour prédire le nombre de défauts du programme que la métrique de complexité de McCabe; en d'autres termes, la relation entre la complexité cyclomatique et le nombre de défauts n'est pas primordial, la taille du programme est le facteur primaire et définitif qui affecte la complexité du programme et ainsi le nombre de défauts générés dans le développement.

- Craddock [49] constate que la complexité cyclomatique est une mesure plus performante que la métrique du nombre de lignes de code pour indiquer le nombre de défauts détectés dans le programme, car la complexité cyclomatique possède un niveau de corrélation plus élevé avec le nombre de défauts détectés dans l'activité d'implantation ou de test logiciel que celle de la métrique du nombre de lignes de code.

- Basé sur l'expérience de Kan [49] et sur l'étude analytique de Troster[57], on constate que la complexité cyclomatique et la densité de défauts sont corrélés de façon significative. En effet, elle est plus fiable comme métrique qui représente la dimension logicielle que le nombre de lignes de code pour prédire la densité de défauts plus précisément.

- Kan [49] note que la mesure de complexité de McCabe originale se fonde sur le nombre cyclomatique en comptant exclusivement le nombre de prédicats des branchements binaires « si-alors-sinon ». Les différents types de prédicats dans la structure de contrôle ne sont pas distingués dans la mesure de complexité, cependant, la boucle « tant...que » est considérée comme un branchement binaire et la sélection contenant N cas est comptée comme $(N-1)$ branchements binaires. En 1992, Lo a réalisé une équation de régression linéaire afin de représenter la relation entre la complexité cyclomatique et le nombre de défauts [49]. Ces différents types de prédicats associés avec leurs propres coefficients sont inclus dans le modèle suivant :

  $$Defects = 0.15 + 0.23 * DoWhile + 0.22 * Select + 0.07 * IfThen$$

  À partir de l'équation, on observe que les prédicats de boucle « *DoWhile* » et de la sélection des cas « *Select* » ont plus d'impact sur les nombres de défauts que la décision binaire « *IfThen* ». La minimisation de l'utilisation des boucles est une manière d'amélioration de la qualité logicielle pour réduire le nombre de défauts durant le développement.

- Plus récemment, dans l'analyse empirique établie par Takahashi [54], il affirme qu'il y une corrélation significative entre la complexité des interrelations entre les unités fonctionnelles[30] d'un sous-système informatique et la densité de défauts dans le test

[30] Unité fonctionnelle est définie par Takahashi[54] comme un programme qui comprend plusieurs modules ou fonctions interdépendantes.

d'un système logiciel. Cependant, une extension de la métrique de complexité cyclomatique a été réalisée dans l'analyse afin de mesurer la complexité des structures de contrôle parmi des unités fonctionnelles; nous la présentons à l'aide de la figure ci-dessous :

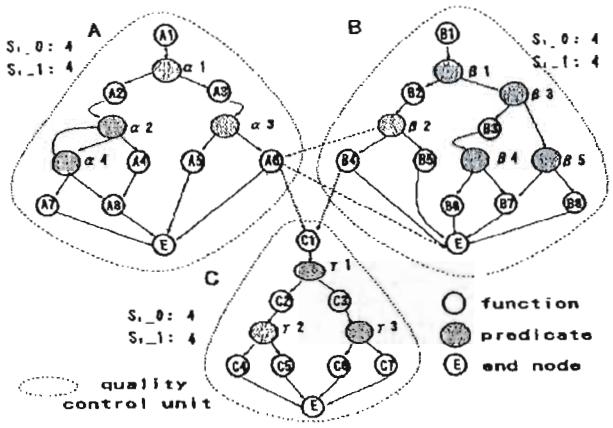

Figure 2.5 Exemple de la démonstration des interrelations entre des unités fonctionnelles d'une application adaptée de Takahashi [54]

Dans la figure 2.5, le nombre cyclomatique à l'intérieur de chaque unité de contrôle (ou unité fonctionnelle) comportant plusieurs fonctions est noté par S_1_0. La portée de la métrique est confiée au nombre de prédicats (+ 1) qui contrôlent seulement l'exécution de fonctions définies à l'intérieur des unités de contrôle. De cette façon, la complexité S_1_0 de l'unité A dans la figure est calculée par $\alpha_2 + \alpha_4 * 2 + 1 = 4$.

La complexité cyclomatique au niveau des interrelations parmi ces trois différentes unités de contrôle A, B, C est mesurée par la métrique d'extension du nombre cyclomatique S_1_1. La mesure dans une unité de contrôle se fait par la sommation du nombre de prédicats qui contrôlent l'exécution de fonctions appartenant à d'autres unités de contrôle et du nombre de prédicats qui proviennent d'autres unités de contrôle en affectant les fonctions définies dans l'unité fonctionnelle de lui-même. On explique par l'exemple que la complexité S_1_1 dans l'unité A est calculée comme $\alpha_1 + \alpha_3 + \beta_1 + \beta_2 = 4$.

Les résultats statistiques sur des études de cas empiriques confirment la validité de la métrique d'extension de complexité cyclomatique. Ils possèdent une corrélation fiable et forte avec la densité de défauts en fournissant une prédiction de la qualité logicielle plus rigoureuse. Takahashi observe également que la complexité cyclomatique dans un module n'est pas corrélée significativement avec la densité de défauts.

2.2.4.2 Les métriques dans la structure de flux de données

Ce type de mesure prend en compte les interactions entre des composants[31] constitués à la conception d'un système et quantifie ces interactions afin d'évaluer globalement la complexité structurelle de la conception du système [40;49]. Les notations de « couplage » et de « cohésion » sont deux critères et concepts principaux adoptés dans les métriques de flux de données. Elles ont étés introduites par Constantine, Steven et Myers en 1974 [30; 40; 49]: le couplage exprime le degré d'interdépendance entre les composants du système. La conception d'un système logiciel expose le couplage global comprenant tous les couplages possibles associés par paires de composants du système; un couplage fort du système implique des difficultés dans la maintenance du logiciel. La cohésion d'un composant réfère à la portée des parties de ce composant exigées pour implanter une seule fonction. Elle est distinguée de plusieurs niveaux dans l'ordre croissant : cohésion coïncidente, association logique, cohésion temporelle, cohésion procédurale, cohésion de communication, cohésion séquentielle et cohésion fonctionnelle. La cohésion coïncidente indique que le composant implante plusieurs fonctions du système qui ne sont pas liées; ceci résulte un système plus complexe et plus difficile à maintenir. En revanche, la cohésion fonctionnelle est la qualité la plus favorable dans la conception du système. Cependant, chaque composant du système implante une seule fonction bien conçue. En résumé, le faible couplage et la forte cohésion apportent plus de fiabilité et plus de facilité durant la construction et la maintenance du système.

Un ensemble de métriques de flux de données est proposé afin de modéliser le couplage et la cohésion du système, tout en fournissant la quantification de leurs degrés pour chaque composant particulier du système. Ces métriques nous permettent de prédire les faiblesses dans la conception du système; ces faiblesses ont des effets négatifs sur la qualité du système en ce qui touche à sa fiabilité et sa facilité d'entretien. Dans ce qui suit, nous introduisons deux métriques essentielles adoptées durant l'activité de conception du système [30;40;49]: la métrique de flux d'information et la métrique de complexité du système.

La mesure de flux d'information

D'après [30;40], Henry et Kafura modélisent la complexité de chaque composant en fonction de son « Fan-In »[32], de son « Fan-Out »[33] et de sa taille en nombre de lignes de code; elle est décrite par la formule suivante :

$M = (L \times fan-in \times fan-out)^2$ où M est la complexité du flux d'information, L dénote la taille du composant. Les mesures « Fan-In » et «Fan-Out » avec de grandes valeurs introduisent la structure du composant sans cohésion, car elles signifient que le présent composant exécute plusieurs fonctions en entraînant plus de flux d'information à travers lui même. Le couplage est quantifié par la complexité de flux d'information M ; la complexité M élevé indique que les composants sont interconnectés fortement et ceci résulte en moins de facilité de maintenance du systèmes.

Fenton [30] a évalué la métrique de flux d'information dans un perspective de la théorie de la mesure. Il note que la multiplication entre le « Fan-In » et « Fan-Out » peut occasionner l'absence de complexité (zéro dans le résultat), car la mesure « Fan-In » ou « Fan-Out » d'un composant est souvent zéro dans la conception du système. Shepperd [30] a adapté des raffinements dans la mesure afin de rendre la comptabilisation du nombre de chemins de flux d'information plus précise. Cependant, les attributs de structure de flux d'information sont caractérisés plus spécifiquement comme la prise en compte de la réutilisation du composant ou des appels récursifs dans la comptabilisation de la mesure, et ignore l'effet du facteur de la taille du composant dans la formule originale, etc. Le perfectionnement de la mesure est validé par le résultat expérimental.

---

[31] Le composant du système est un élément atomique identifié par la décomposition descendante d'un système dans une structure hiérarchique.

[32] La métrique « Fan-In » d'un composant est le nombre de chemins de flux d'information destinés à ce composant, autrement dit, c'est le nombre de fois où le composant est appelé par d'autres composants.

[33] La métrique « Fan-Out » d'un composant représente le nombre d'appels émis par le composant.

La mesure de la complexité du système

En 1970, Card et Glass [49; 53] proposent la mesure de complexité du système global dans une structure hiérarchique. Cette mesure incorpore la complexité du modèle de données au sein de chaque composant et la complexité des interdépendances entre les composants et seulement la mesure « Fan-Out » est prise en compte dans la formule de calcul. L'équation initiale est décrite dans ce qui suit:

$C_t = S_t + D_t$ où la complexité du système est représentée par C_t. La complexité des interdépendances des composants S_t est calculée par la formule $S_t = \sum(f(i)^2)$ où le paramètre $f(i)$ dénote le nombre de « Fan-Out » de chaque composant i du système.

La complexité interne totale des composants du système D_t est calculée par la formule $D_t = \sum^{V(i)}/_{(f(i)+1)}$, où $V(i)$ indique le nombre de variables Entrée/Sortie du composant i et f_i est le nombre de « Fan-Out » du composant. Ceci veut dire que la complexité interne d'un composant est dépendante linéairement du nombre de variables $V(i)$. Elle est corrélée inversement avec le nombre de « Fan-Out », parce que plus de variables Entrée/Sortie dans un composant introduisent plus de fonctions exigées et plus de complexité interne du composant; au contraire, un nombre « Fan-Out » plus grand indique que la fonction du composant est décomposée et implémentée dans les composants des niveaux plus bas dans la conception hiérarchique du système, et la complexité interne du composant est réduite. Finalement, la complexité moyenne du système global C est définie comme $C = (S_t + D_t)/n$ où n est le nombre de composants dans le système. Pour une valeur relative de complexité moyenne inférieure à 26, le système est identifié comme ayant une qualité élevée.

Analyse des métriques dans la structure de flux de données

À partir des revues [30;40;49;53], nous établissons dans ce qui suit nos propres points de vue sur les métriques dans la structure de flux de données :

- Les métriques de flux de donnée sont basées sur la conception du système; cette dernière est produite dans les phases initiales du développement. Ces métriques fournissent des indicateurs quantitatifs pour prédire la qualité du système surtout en terme de sa fiabilité, et de sa facilité de maintenance. Selon les résultats empiriques de Card et Glass, on observe que la complexité du système global est corrélée fortement avec le taux d'erreur durant la construction. En fonction de la mesure, on pourra prévoir des composants enclins à l'erreur, ce qui surviendra durant l'implantation du système, ainsi que l'amélioration de la facilité de maintenance du système.

- Les métriques de flux de données sont des directives quantitatives afin de modéliser la qualité du système classifié en différents niveaux de complexité structurelle sur l'échelle ordinale. Mais la procédure pour établir l'association entre la valeur relative de ces métriques et ces différents niveaux de complexité sur l'échelle ordinale est très subjective en se basant sur des jugements d'expert. On manque ainsi d'évidences expérimentales plus fiables; de plus, la complexité logicielle est influencée par divers facteurs imprévisibles durant le processus de construction comme l'utilisation de différents langages de programmation, l'environnement de développement, le changement des exigences du client ou de conception du système dans la construction et le facteur humain,, etc. Nous en déduisons qu'il est difficile de construire un modèle de classification de la qualité du logiciel de manière fiable et consistante.

2.2.5 Les mesures dans le paradigme orienté objet

De nos jours, le paradigme orienté objet est de plus en plus populaire pour la conception et le développement de logiciel. Il permet de modéliser le système à réaliser en termes d'objets. Ceci est fondamentalement différent de la conception conventionnelle qui élabore le système en vue de ses fonctions, et considère les données et les procédures ou modules fonctionnels séparément dans la structure du système. Pourtant la majeure différence entre l'approche orientée objet et l'approche conventionnelle est la taille des composants procéduraux. De ce fait, les métriques appliquées dans le développement conventionnel comme le nombre de lignes de code, ou la complexité cyclomatique sont inadaptées aux notations orientées objets de classes, d'héritage, d'encapsulation etc. Dans cette section, nous présentons et analysons une suite de métriques orientées objets proposées par Chidamber et Kemerer [59]. Ces mesures spécifiques sont établies afin de mesurer la complexité dans la conception des classes; cette dernière comprend la complexité et la taille des classes du système, l'utilisation d'héritage, le couplage entre classes, la cohésion dans une classe et la collaboration entre classes. Nous les couvrons dans ce qui suit [55; 58; 59].

Le nombre pondéré de méthodes par classe « weighted methods per class --- WMC »

Cette mesure permet de pondérer le nombre des méthodes d'une classe par leurs propres complexités internes; elle est formalisée par WMC = $\sum_{i=1}^{n} C_i$ si la classe définit n méthodes, C_i dénote la complexité individuelle de chaque méthode. En fonction de la validation empirique de la métrique qui est effectuée par Chidamber et Kemerer, le nombre de méthodes et leurs complexités dans une classe permettra de prédire le délai et l'effort exigés dans la construction et la maintenance de cette classe. Plus il y a de méthodes définies dans la classe, plus il y aura d'impact sur leurs classes descendantes qui héritent de toutes les méthodes de la classe parente. Également, une classe avec un grand nombre de méthodes indique qu'elle est plus spécialisée, ce qui réduit la possibilité de réutilisation de la classe dans la construction.

La profondeur de l'arbre d'héritage « Depth In The Tree --- DIT»

Ceci permet de mesurer la longueur du chemin représentée par les niveaux entre une classe et la classe racine dans la hiérarchie d'héritage de la conception du système; dans le cas d'héritages multiples d'une classe, on prend en compte la longueur maximum. Une classe ayant une valeur plus élevée impliquera une complexité plus grande et une certaine difficulté à prédire le comportement de la classe; un arbre d'héritage plus profond suggère non seulement une conception de système plus compliquée, mais aussi plus de réutilisations des méthodes héritées.

Nombre de classes descendantes dans l'arbre d'héritage « Number Of Children --- NOC »

Selon sa définition [59], cette mesure est le nombre des sous classes immédiates d'une classe dans la hiérarchie des classes. On veut aussi mesurer l'influence d'une classe sur la conception du système; en d'autre termes, on quantifie la portée des sous classes impliquées qui héritent des méthodes de la classe parente. Un nombre de classes descendantes plus élevé montrera certains problèmes qualitatifs et de performance dans la conception du système. Ces derniers comprennent la réutilisation de la classe parente par le plus grand nombre de sous classes qui sont liées directement, une abstraction inacceptable de la classe parente et une exigence d'effort augmentée pour tester cette classe.

Le couplage entre des classes d'objets « Coupling Between Objects --- CBO »

La métrique est développée pour mesurer la dépendance entre classes dans la conception du système; le concept de couplage entre deux classes est défini comme l'interaction d'une classe avec une autre par invocation des méthodes ou accès aux variables d'instance définies dans la deuxième classe. La mesure du couplage d'une classe est représentée par le nombre de classes couplées avec celle-ci. Un fort couplage représenté par une grande valeur indique que la classe sera plus complexe et plus difficile à comprendre tout en ayant une réutilisabilité diminuée, moins de facilité de modification et moins de facilité de maintenance. En revanche, un faible couplage de classe est favorable en amenant plus de modularité et en améliorant l'encapsulation de la classe.

Références à une classe « Reference For Class --- RFC »

La mesure est définie comme le nombre de toutes les méthodes accessibles qui peuvent être invoquées et exécutées lors de la réception d'un message; il s'agit d'un ensemble de méthodes comprenant des méthodes appelées à l'interne de la classe, ainsi que des méthodes invoquées dans d'autres classes. Cette mesure est exprimée sous la forme suivante :

$RPC = |RS|$ où RS représente l'ensemble des méthodes impliquées en réponse à un message reçu par la classe; il est défini par la formule suivante : $RS = \{M\} \cup_{all} \{R_i\}$ tels que M représente l'ensemble de toutes les méthodes invoquées dans la classe C, R_i est l'ensemble des méthodes dans les autres classes qui sont appelées par une méthode i de la classe C.

La métrique permet de mesurer non seulement la complexité interne de la classe par le nombre de méthodes invoquées, mais elle quantifie encore les interactions essentielles en considérant des méthodes des autres classes directement appelées par une méthode de la classe. En effet, la mesure tente d'introduire la complexité de la classe en mêlant la complexité dans la classe et celle de ses interrelations avec d'autres classes. Un nombre de méthodes invoquées plus élevé en réponse au message signifie une complexité de classe plus

élevée. Ceci exige également plus d'effort et plus de temps au testeur de logiciel pour comprendre, inspecter et déboguer le programme.

Manque de cohésion des méthodes « Lack Of Cohesion Of Methods--- LOC»

La cohésion dans une classe représente la similarité des méthodes de classe qui possèdent un accès aux mêmes variables d'instance dans la classe. Le degré de similarité est défini comme le nombre de variables d'instance qui sont partagées par une paire de méthodes dans la classe, et est calculé par la formule suivante : $|\sigma()| = \{I_1\} \cap \{I_2\}$ où $|\sigma()|$ est le degré de similarité, les ensembles $\{I_1\}$ et $\{I_2\}$ sont deux ensembles des variables d'instance accédées par les deux méthodes M_1 et M_2.

La métrique du manque de cohésion des méthodes est proposée afin de mesurer la distance entre les méthodes dans une classe. Elle se fait par la soustraction du nombre de méthodes en paires pour lesquelles le degré de similarité est zéro du nombre de méthodes en paires qui sont associées par l'accès aux mêmes variables. Chidamber et Kemerer énoncent qu'une cohésion élevée est favorable dans la conception orientée objet, parce qu'elle tend à promouvoir l'encapsulation dans la classe, ceci apportant une simplicité et une réutilisabilité élevées de la classe. En revanche, la valeur élevée du manque de cohésion représente une faible cohésion dans la classe augmentant sa complexité de conception; en conséquence, ceci amène la possibilité d'avoir plus d'erreurs durant l'implantation de la classe. Chidamber et Kemerer suggèrent que la division de cette classe en plusieurs sous classes avec cohésion plus élevée est une solution possible pour résoudre la faible cohésion de classe.

Analyse des métriques orientées objets

Plusieurs travaux ont été entrepris afin d'évaluer cet ensemble de métriques orientées objets théoriquement et empiriquement; nous les mentionnons ci-dessous [53; 55; 58; 59]:

- Chidamber et Kemerer [59] ont effectué une étude de validation de ces métriques orientées objets à partir de systèmes commerciaux écrits en C++ et Smalltalk. Ils remarquent que les métriques sont utiles durant la construction du système, l'utilisation de la suite de métriques fournissant des indices utiles aux gestionnaires du projet pour évaluer la qualité de la conception du système:

- En utilisant les métriques telles que WMC, DIT et NOC, on pourra identifier la quantité de classes définies au niveau de la racine dans la conception du système, un nombre de classes excessif signifie peu d'héritage dans le système et produit de nombreuses méthodes dans le premier niveau de la conception du système; ça réduit essentiellement la réutilisation des méthodes. D'un autre coté, les métriques RFC et CBO vérifient s'il y a des interconnexions inadéquates entre des classes dans la conception. Rosenberg et al. [53] ont proposé une directive quantitative en spécifiant les seuils qualitatifs dans la conception orientée objet; cependant, si les métriques orientées objets atteignent leurs propres valeurs de seuil, la modification devra être faite dans la conception de classe. On présente ci-dessous les critères numériques définis pour chaque métrique.

  Réponse pour une classe « RFC » = 100

  Réponse pour une classe = 5 fois le nombre de méthodes dans la classe

  Couplage entre objets « CBO » = 5

  Nombre pondéré de méthodes par classe « WMC » = 100

  Nombre de méthodes = 40

- Dans un contexte de gestion de développement, la suite de mesures orientées objet peut être utilisée afin d'identifier les parties plus complexes dans la conception du système qui exigent plus d'effort pour les implanter et les tester. Ceci offre une aide aux allocations de ressources de développement afin d'avoir une utilisation des ressources plus efficace.

- Egalement les métriques orientés objets sont applicables pour surveiller le processus de construction du système. Cependant, la consistance entre la conception architecturale du système et son implantation est évaluée durant la construction. En analysant la stabilité des valeurs des métriques comme WMC, NOC et DIT, les gestionnaires pourront déterminer s'il y a des modifications significatives apportées à la structure du système durant l'implantation. Les changements de valeurs des métriques RFC et CBO occasionnés par de nouvelles mises à jours des couplages ou des interrelations des classes du système permettent de suivre le processus d'évolution du système orienté objet de façon continue.

- D'après la discussion critique de Churcher et Shepperd [58], il y a des ambiguïtés dans les définitions des métriques orientées objets de Chidamber et Kemerer. Selon une étude empirique d'un système industriel en C++, Churcher et Shepperd révèlent que ces ambiguïtés provoquent des interprétations des utilisations des métriques orientées objets qui influencent significativement les résultats empiriques.

- Pour mesurer le nombre pondéré de méthodes par classe « WMC », selon la définition d'une méthode donnée par Chidamber et Kemerer, le nombre de méthodes d'une classe ne clarifie pas l'inclusion des méthodes héritées de sa classe parente. En d'autres termes, la définition imprécise de méthodes apporte des difficultés aux testeurs pour compter le nombre de méthodes du système; ce dernier est significativement influencé par l'héritage dans la conception orientée objet.

- Également, dans la mesure du couplage entre classes, cette notion n'est pas définie spécifiquement. Lorsqu'un appel d'une classe invoque une méthode héritée d'une autre classe, il n'est pas clair que le couplage doive référer au lien direct entre la première classe et la deuxième classe ou au lien indirect entre la première classe et la classe parente de la deuxième classe.

CHAPITRE III

MESURE DE LA PROGRESSION DE LA CONSTRUCTION DE LOGICIEL

Dans ce chapitre, nous faisons une présentation et une étude analytique des métriques appliquées au domaine du contrôle de projet pour mesurer l'avancement du projet durant la construction. En premier lieu, nous commençons par montrer la dépendance mutuelle qui existe entre la mesure de la progression et le processus de contrôle du développement. Puis nous adoptons le cadre de gestion de projet modélisé par Ho Leung Tsoi [60], afin d'expliquer plus en détail l'utilisation de ces métriques de progression ainsi que leur rôle pour aider le processus de contrôle de la construction. Par la suite, un ensemble de métriques et de méthodologies adaptables à la mesure de progression de la construction sera introduit et analysé. Finalement, nous donnons quelques conclusions avec des analyses et des suggestions pour les mesures de progression en vue d'améliorer la qualité du processus de construction du système logiciel.

3.1 Relation entre mesure de progression et contrôle du développement du projet

Selon l'idée de Colin Kirsopp [1], la mesure logicielle est réciproquement dépendante du processus de développement. Ce dernier effectue la définition et la validation théorique des métriques, détermine le moment d'utilisation des métriques ainsi que la manière d'interpréter les résultats des mesures. Kirsopp affirme que la mesure logicielle doit être considérée comme une activité indispensable dans le processus de développement du projet. En subdivisant le processus de développement du projet en plusieurs sous étapes de haut niveau,

Kirsopp réalise un schéma d'intégration du programme de mesure dans les sous étapes associées au processus de développement:

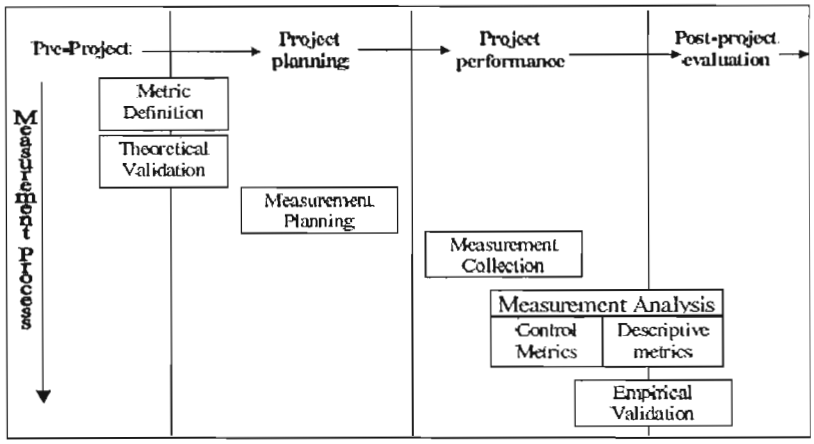

Figure 3.1 Intégration du programme de mesure dans le processus de développement adapté de Kirsopp [1]

À partir de cette figure, le programme de mesure logicielle comprend quatre activités principales exposées comme suit [1]:

- la définition des métriques candidates et la validation théorique de ces métriques selon les besoins du processus de développement dans les étapes préliminaires du développement;

- la planification de la mesure où les gestionnaires sélectionnent des métriques adéquates et déterminent quand elles seront utilisées au cours de la construction;

- la collection et l'analyse des données de mesure; cette activité de mesure doit être mise en oeuvre dans l'étape de construction du projet. Selon les indications de ces métriques, les décisions d'arbitrage seront prises par les gestionnaires au cours du développement du projet afin d'identifier et de remédier aux problèmes du développement au bon moment. Cependant, deux types de métriques classées en vue

de leur utilisation sont mises en places: la métrique de contrôle[34] et la métrique descriptive[35];

- la validation empirique des métriques est un processus continu qui survient lorsque les résultats des mesures utilisées sont disponibles durant la construction ou après la fin du projet.

Selon le point de vue de Kirsopp [1], on constate qu'il y a interdépendance mutuelle entre la mesure logicielle et le processus de développement du projet, particulièrement dans l'étape de construction du projet. L'implantation du projet sera contrôlée et conduite par l'activité de mesure.

Dans ce qui suit, on introduit un cadre de gestion de projet établi par Ho Leung Tsoi [60]; ce cadre adapte les activités du programme de mesure afin de rendre la gestion du projet plus fiable. Ho Leung Tsoi vise à résoudre les problèmes souvent rencontrés dans le développement en ce qui concerne le surcoût du développement du projet et le retard pour livrer le produit final. Les activités de mesure dans ce cadre de gestion sont présentées par la figure 3.2; elles sont organisées surtout dans la phase de planification et dans la phase d'implantation au cours du processus de développement [60]:

- La phase de planification comprend les activités citées suivantes :
    - D'abord, on doit ordonnancer les tâches du projet et estimer leurs tailles afin de prédéterminer leur charge d'implantation;
    - Ensuite l'évaluation des facteurs de changement sera faite. Elle permet d'identifier le risque et l'incertitude associés au processus de construction et d'en prédire les changements possibles durant l'implantation. Ces valeurs incluent les changements apportés dans l'analyse des besoins et dans la conception, dans la technologie de développement et dans la main-d'oeuvre.

---

[34] La métrique de contrôle est la mesure qui apporte une assistance aux gestionnaires dans leurs prises de décisions. L'analyse de ce genre de métrique demande un jugement explicite; la mesure est parfois calculée en combinant plusieurs métriques descriptives selon des formules. Par exemple, la productivité du développeur est améliorée de 10% en comparant la productivité réelle du mois et celle du mois précédent.

[35] La métrique descriptive ne joue pas un rôle dans la décision du contrôle du projet. Elle tente de décrire la valeur réelle en fournissant une connaissance immédiate de la situation réelle dans la construction de projet. Elle est utilisée comme base quantitative pour la comparaison afin de découvrir la tendance du développement, etc. Par exemple : la productivité réelle est de 12 lignes de code par jour.

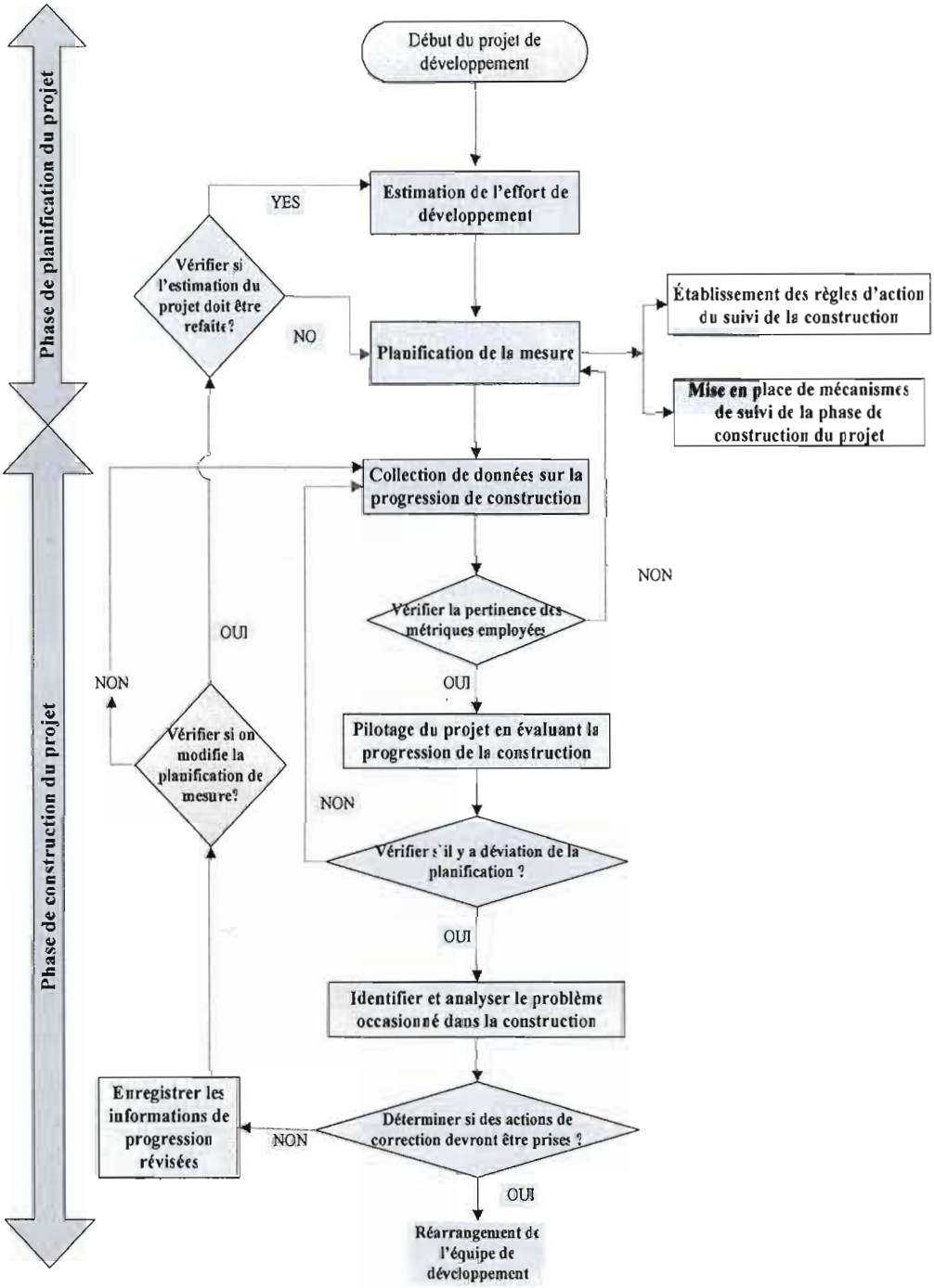

Figure 3.2 Cadre de gestion de la phase de construction selon Lo Heung Tsoi [60]

- Ces changements sont des facteurs cruciaux affectant la progression de la construction; de l'autre coté, en analysant les résultats de l'évaluation, les gestionnaires pourront établir des mécanismes de suivi de progression convenables afin de mieux contrôler l'implantation du projet;

- Finalement dans cette phase, la planification de la mesure est réalisée; elle comporte deux activités principales :

  L'établissement de règles d'action pour suivre l'implantation du projet et la réalisation des mécanismes de suivi de la progression de la construction. La première activité identifie l'objectif du programme de mesure et sélectionne les métriques appropriées pour mesurer le coût, la productivité et la performance ainsi que les fréquences de leur utilisation durant le processus d'implantation. On doit également établir les jalons du projet, ceux qui servent comme indicateurs de performance en montrant la progression de la construction.

  Mise en place de mécanismes de suivi pour assembler les informations des métriques utilisées et analyser les écarts avec la planification du projet, tout en évaluant la progression du développement. Deux types de mécanismes de suivi sont fréquemment adoptés dans le développement: le mécanisme de suivi par jalons et le mécanisme de suivi par points continus. Le premier tente d'évaluer la progression du développement en vue de finir une phase spécifiée durant le cycle de vie du développement du projet. Pour cela, les jalons sont mis en place à la fin de chaque phase du développement. Le suivi par points continus établit un ou plusieurs jalons en utilisant une seule métrique dans une phase spécifique, afin de mesurer la progression du développement de la phase spécifiée.

- Durant la phase d'implantation, le cadre du contrôle de projet permet de garantir un suivi fiable pour mesurer précisément la progression de l'implantation, et prendre les bonnes décisions en cas de déviation de la planification. Il comprend les activités essentielles suivantes :

  - Le suivi de la performance de construction assemble des types d'informations sur la performance d'implantation en termes de productivité de l'équipe et de la progression réelle du projet en termes de taille, de délai et de qualité. Ceci aide à la prise de décision des gestionnaires en examinant la progression et la performance de la construction avec des méthodologies telles que le rapport d'analyse, qui compare les données réalisées et celles estimées lors de la planification du projet, le groupement des différentes équipes de développement, la supervision de la productivité d'une équipe, etc. Cette activité est adaptée continuellement au cours du développement jusqu'à ce que le produit logiciel soit livré au client.

  - Le pilotage du projet est l'élément central dans la gestion de projet; il faut comparer régulièrement ce qui est réalisé durant la construction et la

planification prévisionnelle pour mesurer précisément la progression. Le processus de construction est dirigé par l'activité afin d'atteindre les objectifs qualitatifs et quantitatifs. Cependant, les jalons planifiés sont employés comme pointages intermédiaires pour comparaison avec l'ensemble des données de progression qui sont assemblées par le mécanisme de suivi. Ceci révèle la situation réelle du projet et donne une alerte aux gestionnaires s'il y a des écarts entre la valeur obtenue et celle de la planification.

- À la fin, l'évaluation des résultats de pilotage sera faite par les gestionnaires du projet. Ils tentent d'identifier les problèmes survenus et d'anticiper les impacts que ces problèmes auront dans la future implantation. En fonction de cette évaluation, les gestionnaires pourront prendre la bonne décision pour remédier aux problèmes apparus dans la construction. Également, les gestionnaires devront réviser les mesures de progression ou le mécanisme de suivi établi et la planification du projet estimé. Cependant, ils revérifient la convenance et l'utilité des mesures déjà utilisées et adoptent de nouvelles mesures ou méthodes si nécessaire; de cette manière, le processus de construction est amélioré continuellement en termes de qualité et de performance.

3.2 Conclusion pour la relation entre la mesure de progression et le contrôle du projet

Selon la théorie de Kirsopp [1] et le cadre de gestion de développement proposée par Lo Heung Tsoi [60] on conclut que la mesure de progression est une activité essentielle dans le contrôle de la construction dans le projet. Comme la construction dans le projet est influencée par divers changements imprévisibles, ceci amène souvent un problème de surcharge de la construction dans le projet en termes de surcoût et de retard. En adoptant la mesure de la progression et son mécanisme de suivi dans le cadre du contrôle de la construction, d'une part, on peut poursuivre et superviser le processus d'implantation de manière fiable tout en procurant des informations sur l'avancement du projet aux gestionnaires; d'autre part, avec l'ensemble des informations de progression obtenues au cours du développement, les gestionnaires du projet pourront mieux comprendre et identifier les problèmes apparus dans la construction; ils disposeront des solutions adéquates et proposeront ainsi des ajustements dans la planification du projet afin d'améliorer la qualité et l'efficacité du processus d'implantation. Dans la section suivante, nous allons présenter un ensemble de métriques et de méthodologies adoptées pour mesurer l'avancement de la construction selon la taille, la qualité et la durée.

3.3 Revue des mesures pour la progression de la construction

Un ensemble de mécanismes de suivi et de métriques a été mis en place dans le pilotage du projet afin d'évaluer la progression et la performance de la construction dans le projet en termes de coût du processus, de durée et de qualité du processus de construction. L'utilisation de ces mesures a pour objectif de maîtriser la construction dans le projet tout au long de son déroulement; cependant, les mesures de progression sont prises au cours de la construction comme étant des indicateurs d'avancement et d'efficacité du processus de développement. Elles sont regroupées et présentées sous la forme d'un tableau de bord de pilotage; ce tableau sera analysé par les gestionnaires et permet d'obtenir des renseignements effectifs sur le statut réel du projet; les résultats de l'analyse aident les gestionnaires à prendre des décisions ou faire des jugements pertinents et convenables. Ils visent à garantir l'atteinte des cibles du projet en aidant à remédier aux problèmes dus aux dérives de planification, à anticiper les changements à accomplir et à prévenir les défauts ou les risques potentiels au cours de la construction dans le projet de développement.

Selon des revues [45; 49; 53; 61], ces critères de suivi et de métriques adaptées dans les critères se composent des cinq types décrits ci-dessous. Une fois ces mesures introduites, nous analyserons chaque type de mesure en détail dans les sous-sections 3.3.1; 3.3.2; 3.3.3; 3.3.4 qui suivront.

- 3.3.1 : Mesurer la progression au moyen des jalons du projet [45; 53]; ce type de mesure suit le processus de construction en vérifiant les jalons établis basés sur les temps prévus . Les jalons servent de postes de contrôles associés aux échéances de livraison d'objets intermédiaires achevés dans chaque phase d'un cycle de vie de développement.

- 3.3.2 : Mesurer la progression selon la taille du projet en nombre de lignes de code [49]; le suivi de la taille du projet au cours de la construction présente de multiples utilisations significatives dans le contrôle du projet; elles sont citées dans ce qui suit :

    - L'indicateur de changement d'exigences du client ou de conception du projet provoque souvent un changement de taille évident au cours de la construction;

- La détermination des efforts dédiés aux modules réalisés en utilisant les modèles de régression qui considèrent la taille du projet comme le paramètre d'entrée principal;

- L'utilisation de la taille du projet comme une covariable dans la mesure de la densité de défauts; cette mesure est calculée par le nombre de défauts détecté par unité de code produit, afin de refléter la qualité réelle du code construit au cours du projet;

- Le suivi continu du volume de code source complété et intégré dans la bibliothèque globale du système permet d'établir un schéma référentiel lors de l'achèvement du projet. Le référentiel représente la tendance cumulative de la taille du projet au fur et à mesure de l'intégration du code[36]. En utilisant le schéma établi, les gestionnaires pourront réaliser une planification plus précise pour la progression de l'intégration du code, dans le but de maîtriser la construction d'autre projets similaires de manière plus efficace.

- 3.3.3 : Mesurer la progression via l'utilisation de métriques de processus; ces métriques jouent un rôle effectif dans l'objectif de faciliter le contrôle de la construction, la planification et l'intervention des gestionnaires, tout en assurant la qualité du produit logiciel et en améliorant la performance du processus de test. L'utilisation des métriques de processus est décrite dans les deux démarches suivantes [49; 53]:

  - Le suivi de la progression des activités de test rigoureux du système[37]. Deux métriques sont calculées : le nombre de jeux de tests exécutés par semaine et le nombre de jeux de tests réussis par semaine. Elles sont suivies et pilotées avec le plan de test établi tout en avançant l'activité de test logiciel;

  - Le contrôle de la qualité du logiciel en utilisant les métriques de qualité du processus logiciel comme le nombre de défauts détectés tout au long du cycle de vie de développement, l'efficacité de la suppression des erreurs et la densité de défauts, etc. D'une part, ce sont des indicateurs significatifs

---

[36] L'intégration de code [49; 53] est l'activité fondamentale et coûteuse durant le processus de développement du projet. Elle comprend trois activités essentielles telles que l'implémentation du code source des modules, les tests unitaires ainsi que l'intégration du code réalisé dans la bibliothèque du projet qui est disponible pour le test formel du système global.

[37] Le test rigoureux du système [49] amène l'inspection globale du système et la correction d'erreurs après la phase d'intégration du code.

pour suivre le statut de la performance du processus et de la qualité du produit logiciel de façon répétitive lors du test rigoureux du système. D'autre part, les gestionnaires comparent les valeurs réelles de qualité avec celles planifiées dans un référentiel qualitatif; ce référentiel est établi basé sur des projets similaires et achevés, qui sont souvent représentés par de courbes de référence dans un tableau de bord. Ensuite, les gestionnaires pourront interpréter les déviations dans un perspective de qualité et prendre les bonnes décisions; ces dernières demandent de juger si la fiabilité et la stabilité actuelle du produit sont conformes aux exigences, de remédier aux problèmes en cas de détérioration de la qualité et d'estimer le nombre de défauts subsistants à réparer, etc. Le majeur avantage de la démarche réside dans le fait qu'on peut détecter les erreurs du code et les corriger le plus tôt possible au cours de la construction afin de garantir la facilité de maintenance du système produit et de réduire l'effort de maintenance du système.

- 3.3.4 : Mesurer la progression par adoption de l'approche EVA[38][45; 62]. C'est une méthodologie efficace pour la gestion de la construction permettant de suivre l'avancement du projet en termes de coût et d'échéancier;, plusieurs types de métriques sont utilisés de manière coopérative tout en reflétant la performance et la progression réelle du projet. Ils sont cités dans ce qui suit :

    - Un ensemble de jalons dans les échéanciers prévus sont établis tout au long du cycle de vie du développement. Des coûts budgétaires du travail prévu (BCWS[39]) sont associés à chaque jalon; ils sont convertis et représentés par le chiffre d'affaire en dollars;

    - Au cours de la construction, la progression de la construction est suivie par la métrique des coûts budgétaires du travail effectué (BCWP [40]) qui représente la quantité du travail faite en dollars et la métrique de coûts réels du travail effectué qui dénote les coûts dépensés pour la complétion du travail en dollars (ACWP [41]);

    - À la fin, deux indices de performance sont également fournis et évalués en aidant au contrôle de la construction : l'indice de performance en coût

---

[38] Le terme « EVA » est l'abréviation de « Earned Value Analysis »

[39] Le terme « BCWS » est l'abréviation de « Budget Cost of Work Schedule »

[40] Le terme « BCWP » est l'abréviation de « Budget Cost of Work Performed »

[41] Le terme « ACWP » est l'abréviation de « Actual Cost of Work Performed »

représente le rapport entre le coût planifié et celui consommé par le développement. Il est calculé par la division BCWS / ACWP; une valeur de cet indice supérieure à 1 signifie que le coût réel est inférieur à celui prévu; pour l'indice de performance de la construction en temps représenté par le rapport BCWS / BCWP, une valeur supérieure à 1 indique que le projet avance plus vite que prévu.

En fonction des valeurs obtenues pour ces mesures au fur et à mesure du développement, les gestionnaires pourront avoir les renseignements sur l'avancement du projet en aspect de coût et de temps, permettant d'évaluer la faisabilité du projet le plus tôt possible, ainsi que d'anticiper et de prévenir les facteurs de risques potentiels qui provoqueront des surcoûts ou des retards dans la future construction.

3.3.1 Mesurer la progression par le jalonnement

L'activité de jalonnement permet de planifier le développement du projet dans le temps et de conduire l'avancement du projet en respectant des échéances. Dans le planning du projet, une fois le projet décomposé en ensemble de tâches ordonnancées et une fois les ressources du projet affectées à ces tâches, les gestionnaires doivent établir les jalons des échéances pour ces tâches. Les jalons clefs se définissent comme une suite d'objectifs pour objets intermédiaires produits par chaque phase du développement; ces objectifs sont les résultats attendus de chaque phase de développement comme le fonctionnement, le coût budgété et la qualité d'un objet livrable avec une échéance fixée; en d'autres termes, les jalons clefs sont conçus pour valider si les objectifs sont atteints par l'objet généré de chaque phase.

Au cours du déroulement du projet, la dérive entre l'objectif réel atteint et celui qui est prévu au départ du projet occasionne la révision de la planification du projet par des ajustements comme une réassignation des ressources aux tâches pour corriger le tir en reportant les tâches restantes à plus tard. Ceci rend le contrôle du projet plus performant en assurant la complétion du produit final.

Le mécanisme de suivi de la progression par le jalonnement utilise souvent l'aide d'un outil de gestion, le diagramme de Gantt, qui permet de représenter graphiquement le planning du projet par des barres. La figure ci-dessous donne un exemple d'utilisation de diagramme de Gantt comprenant les jalons clefs représentés par des losanges dans chaque phase du

développement. Les tâches ordonnancées sont représentées par des barres dont les longueurs représentent les durées estimées et les échéances des tâches.

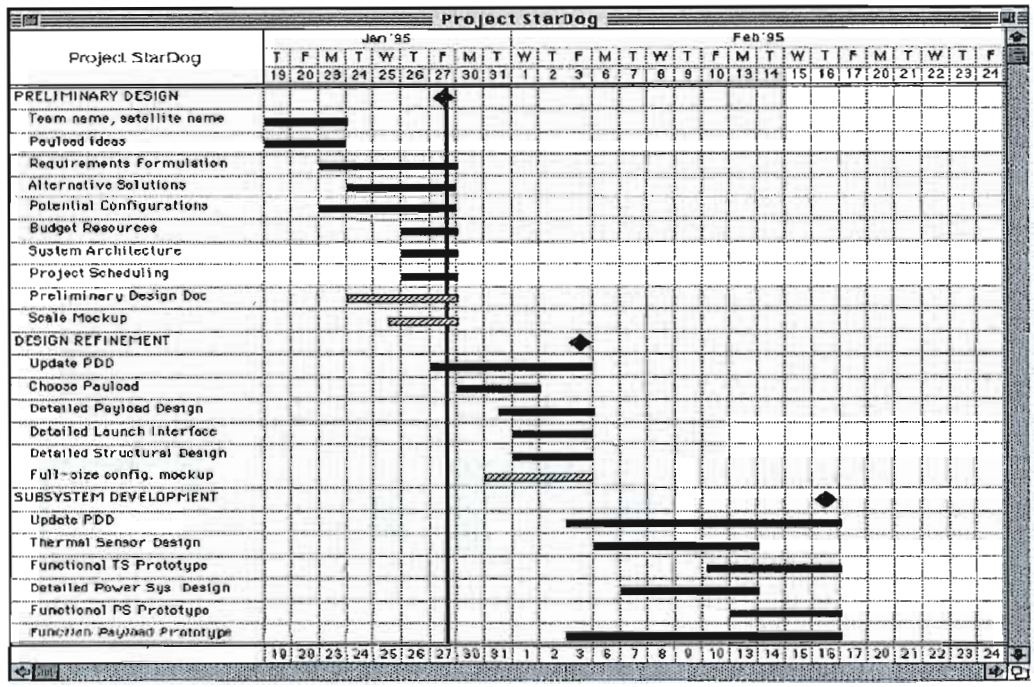

Figure 3.3 Exemple de diagramme de Gantt appliquant le jalonnement adapté de Laird et Brennan [53]

Analyse de l'approche

D'après Laird et Brennan [53], le suivi de la progression par jalonnement est la méthodologie efficace pour contrôler le développement du projet. Des jalons prédéterminés sont adoptés comme objectifs intermédiaires afin de piloter le projet en traversant le cycle de vie du projet. Laird et Brennan ont défini un ensemble de critères pour élaborer les jalons; ces critères sont décrits dans ce qui suit pour définir efficacement les jalons dans le planning du projet :

- Chaque jalon établi pour chaque tâche identifiée dans le planning doit être supervisé par un gestionnaire donné qui devra interpréter le résultat du pilotage en comparant la valeur réelle et celle du jalon et prendre ensuite la bonne décision;
- Les objectifs définis par le jalon doivent être précis et mesurables, afin de déterminer plus explicitement si l'exécution de la tâche est conforme aux objectifs;
- Les jalons définis doivent être faisables; cependant, ils peuvent être basés sur l'expérience de projets similaires achevés dans lesquels on peut avoir une meilleure correspondance entre l'exigence client et les ressources disponibles;
- Les jalons établis doivent être associés à une échéance raisonnable.

3.3.2 Mesurer la progression par le suivi de la taille du projet

Ce mécanisme de suivi fournit une supervision continuelle de l'évolution de la taille du projet dans le temps, au cours de l'activité d'intégration du code. L'utilisation d'un schéma référentiel de tendance de la taille du projet est une activité indispensable et effective pour la gestion du projet. On l'utilise avec pour objectif d'établir le planning de l'intégration du code dans la construction, de piloter le projet en mesurant la progression de la construction, de refléter le changement de la taille du projet durant le développement, etc. Dans la figure 3.4 suivante, Kan [49] nous présente un exemple de schéma d'intégration du code du projet :

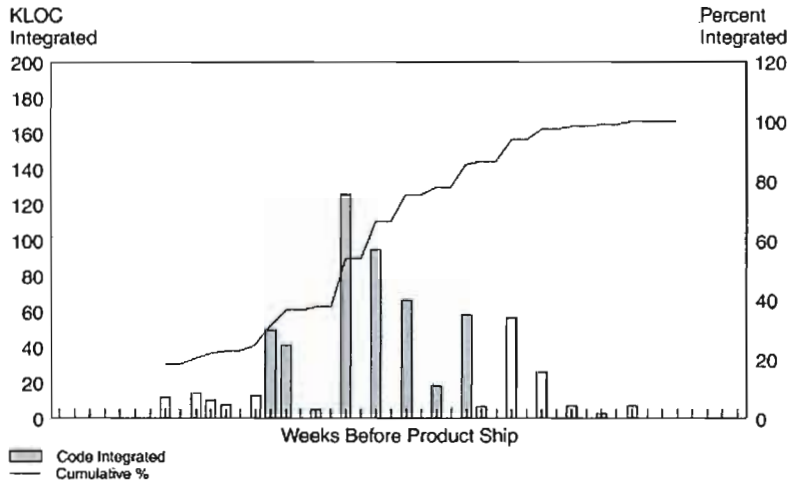

Figure 3.4 Le schéma référentiel d'intégration de code d'un système logiciel donné par Kan [49]

Dans la figure 3.4, le mécanisme de suivi utilise deux métriques de taille du projet: la mesure en unités de code KLOC (kilo lignes de code) exprimée via le premier axe des Y et le pourcentage d'intégration du code représenté par le second axe des Y, les barres verticales décrivant le suivi continu sur le volume du code qui est complété et intégré dans la bibliothèque du système par unité de temps durant la construction; la courbe d'intégration du code en forme de « S » introduit le pourcentage cumulatif d'intégration du code du projet.

Analyse du mécanisme de suivi

Kan [49] a constaté les rôles fournis par le schéma de taille dans le contrôle du projet, qui sont décrits dans les points suivants.

- Les schémas référentiels des projets achevés sont des indices significatifs pour établir la courbe du plan d'intégration du code d'autres projets lors du début du développement. Cependant, des expériences sur des cédules planifiées pour les activités clefs de la construction comme l'achèvement de la conception du système, l'accomplissement de l'intégration du code et la livraison du produit logiciel une fois les tests rigoureux complétés, permettent aux gestionnaires d'établir les jalons d'échéance pour superviser la progression au cours de la construction.

- En comparant et analysant la courbe réelle du plan d'intégration du code établi au début du projet avec celles établies pour des projets complétés, les gestionnaires pourront évaluer la faisabilité du projet actuel en analysant la différence statistique entre ces schémas, anticiper les risques potentiels qui seront rencontrés dans la mise en oeuvre du projet et enfin effectuer les ajustement pertinents au planning du projet afin d'y remédier, tout en améliorant la qualité du processus de construction. Kan illustre cette sorte d'utilisation dans la figure 3.5, expliquée dans ce qui suit :

    - Les produits X, Y et A sont achevés et possèdent leur propres courbes d'intégration du code en forme de « S »; les pourcentages d'intégration augmentent souplement du bas vers le haut, ce qui démontre que les modules complétés sont intégrés à la bibliothèque au fur et à mesure de la construction, les objectifs des projets en termes de délai ou de qualité ont été respectés;

    - Dans le cas contraire, la courbe du plan d'intégration dans le présent projet B indique une croissance à pic vers le haut. Un tel schéma révèle que les modules de gros volume seront complétés et intégrés dans la bibliothèque tous ensemble vers la fin de l'intégration du code. Celle-ci présente des risques potentiels : le retard des tests rigoureux du système, la possibilité d'avoir plus de défauts détectés dans le code intégré qui cause une détérioration de la qualité du produit logiciel;

    - D'après l'étude statistique qui compare le projet B avec les autres projets, la différence est significative du point de vue statistique; c'est à dire que les objectifs déterminés par les chefs de projet ne sont pas réalisables et sont inadéquats en termes du temps prévu, de la qualité du produit et de la taille du projet. Les gestionnaires du projet devront revoir le planning du projet en définissant de nouveaux jalons plus raisonnables. En revanche, si la différence n'est pas évidente entre le projet B et les autres, des actions de corrections pertinentes seront adoptées par les gestionnaires afin d'améliorer la qualité du processus. La suite d'actions correctives comporte le redécoupage des gros modules en morceaux plus petits qui peuvent être complétés et intégrés plus tôt durant la construction, l'analyse de la taille de chaque tâche décomposée et la réaffectation des ressources humaines disponibles aux tâches difficiles afin d'équilibrer la charge de développement;

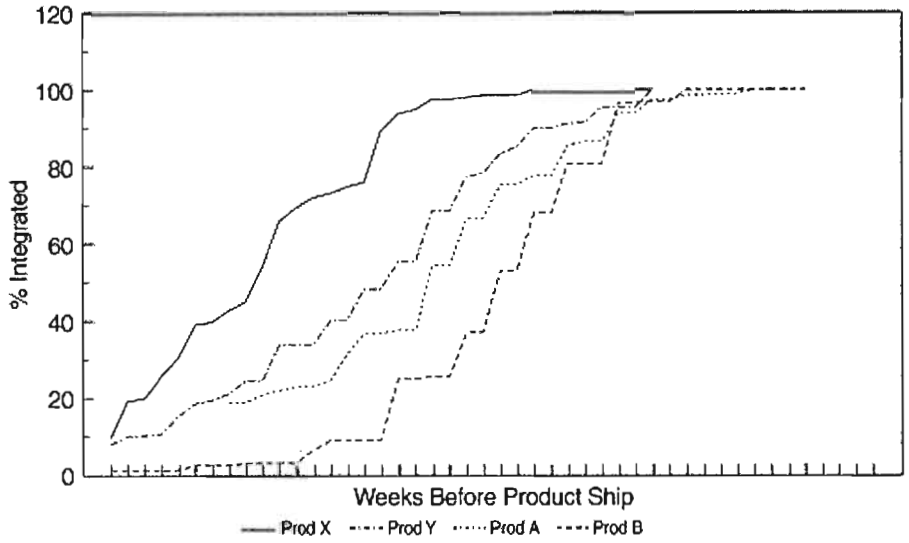

Figure 3.5 Les schémas référentiels de quatre projets X, Y, A qui sont complétés et le projet B au début de son développement présentés par Kan [49]

Dans les courbes du plan d'intégration du code montrées ci-dessus, la métrique du pourcentage cumulatif d'intégration du code est adoptée afin de décrire la progression graduelle de l'intégration du code. Laird et Brennan [53] ont noté que cette mesure est moins pertinente et moins fiable que l'adoption de la mesure du nombre de lignes de code pour montrer la progression de la construction, parce que la courbe du plan d'intégration utilise la mesure du pourcentage d'intégration et est basée sur l'opinion subjective des experts. Ensuite, cette mesure est définie de manière imprécise et implicite; par exemple, si le projet est complété à 80%, une telle mesure fournit souvent une information trompeuse et donne de mauvaises interprétations sur l'état réel d'avancement du projet.

3.3.3 Mesurer la progression en adaptant les métriques de processus

Dans cette démarche, on présente en premier les métriques appliquées du processus logiciel pour mesurer la progression de l'activité de tests rigoureux du projet qui causent toujours la livraison tardive du produit logiciel au client. Puis les métriques de qualité du processus seront présentées et analysées; ces mesures permettent de suivre le statut réel de la qualité du projet afin d'évaluer le progrès vers les objectifs qualitatifs préétablis : également, elles

servent comme indicateurs significatifs pour mesurer l'efficacité du processus de développement et la productivité de l'équipe de développement.

3.3.3.1 Le suivi de la progression de l'activité de tests rigoureux du système

Cette approche utilise un plan de tests pour suivre et évaluer la progression du processus de tests du système. Cependant, les jalons d'avancement des tests sont définis dans le plan de test, et peuvent être comparés avec les valeurs réelles afin de déterminer si l'activité prend du retard et aussi garantir l'efficacité du processus de tests du logiciel. Trois métriques sont souvent utilisées afin d'observer explicitement la progression des tests du système.

- Le plan de tests décrit la progression espérée des tests du logiciel avec le nombre cumulatif de jeux de test à faire durant le test;
- Le nombre de jeux de test exécutés par unité de temps;
- Le nombre de jeux de test exécutés avec succès par unité de temps;

La figure 3.6 [53] ci-dessous montre un exemple de suivi de la progression des tests rigoureux d'un système. Le plan de test est représenté par les deux courbes des schémas référentiels qui indiquent la limite supérieure et la limite inférieure du nombre de jeux de test réussis. Ces courbes sont réalisées à partir des projets complétés; si le nombre de jeux de test complétés réel n'est pas dans cette gamme prédéfinie, des ajustements doivent être apportés pour affecter l'effort des tests efficacement. Un tel plan de tests s'assure que l'activité de test est effectuée de manière performante en procurant des informations sur l'avancement et la performance des tests du système dans les délais.

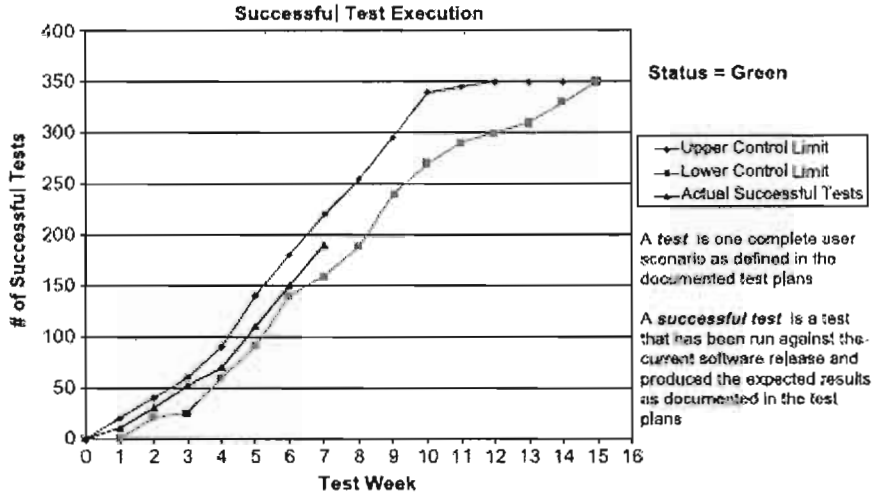

Figure 3.6 Le tableau de bord de la progression de l'activité de test logiciel [53]

3.3.3.2 Contrôle de qualité logicielle par des métriques de qualité du processus

La démarche de la qualité logicielle comprend des activités concernant la définition du plan qualitatif du projet et l'anticipation des risques éventuels au plus tôt dans le projet, la vérification régulière du statut du projet dans une perspective de qualité tout au long du processus des tests rigoureux du système. Les métriques de qualité du processus sont déployées dans les tests du projet comme le nombre d'apparitions de défauts, la densité de défauts et la sévérité des défauts, etc. Ces mesures apportent une aide directe aux gestionnaires du projet pour effectuer les prises d'actions correctives en conformité avec les besoins des clients. L'évaluation des valeurs de ces métriques apporte d'une part l'assurance du bon fonctionnement du produit livrable conformément aux exigences des clients, et d'autre part elle permet d'améliorer continuellement l'efficacité du processus de tests du logiciel et la qualité intrinsèque du produit logiciel en terme de sa fiabilité, sa convivialité et sa facilité de maintenance. D'après des revues [45; 49; 53; 61; 62], dans ce qui suit on cite les métriques de qualité du processus et leur influence sur le déroulement du projet.

Schéma référentiel de l'apparition de défauts

Ce schéma référentiel est construit dans la planification du projet; il est basé sur les données historiques assemblées à partir d'un projet similaire terminé ou sur la mise en production précédente du projet [49]. Plusieurs métriques sont utilisées et suivies dans les tests rigoureux du système afin d'établir les schémas référentiels :

- Suivre le nombre total d'apparitions de défauts détectées lors des tests du système. La figure 3.7 ci-dessous montre les schémas référentiels entre trois mises en production d'un projet. On voit que ces courbes augmentent tout au long des tests du système et tendent à se stabiliser vers un petit nombre de défauts avant la livraison du produit logiciel. Dans la figure, le produit de mise en production A possède plus de qualité que les deux autres mises en production B et C, car la fin de la courbe référentielle A présente moins d'apparitions de défauts, ce qui signifie que le produit logiciel A aura plus de qualité.

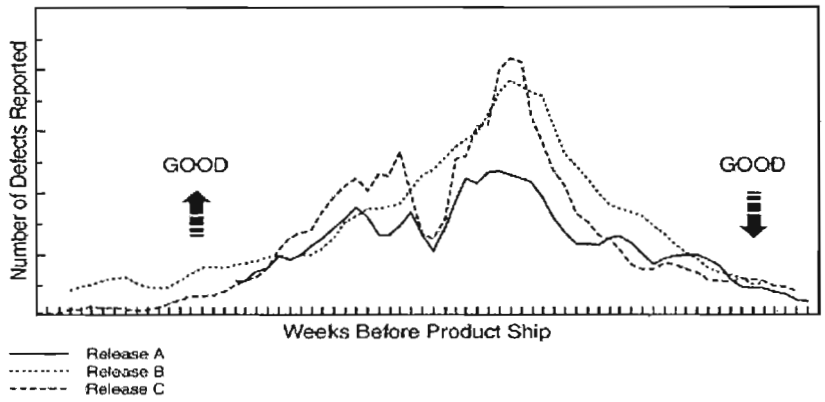

Figure 3.7 Les schémas référentiels d'apparitions de défaut dans les différentes mises en production d'un projet logiciel [49]

- Suivre la sévérité des défauts : chaque défaut est classé selon quatre niveaux d'une échelle de sévérité. Les niveaux 1 et 2 représentent les défauts plus sévères, durant les tests du système; les métriques de pourcentage entre les défauts plus sévères et les défauts totaux sont cumulées afin d'établir le schéma référentiel. La figure 3.8 suivante montre que le pourcentage de défauts sévères augmente avec l'avancement des tests du système; la courbe référentielle de mise en production B présente un grave problème de qualité de produit et des actions correctives doivent être mises en place, parce que le pourcentage de défauts sévères est plus élevé que les deux autres avant la livraison du produit au client.

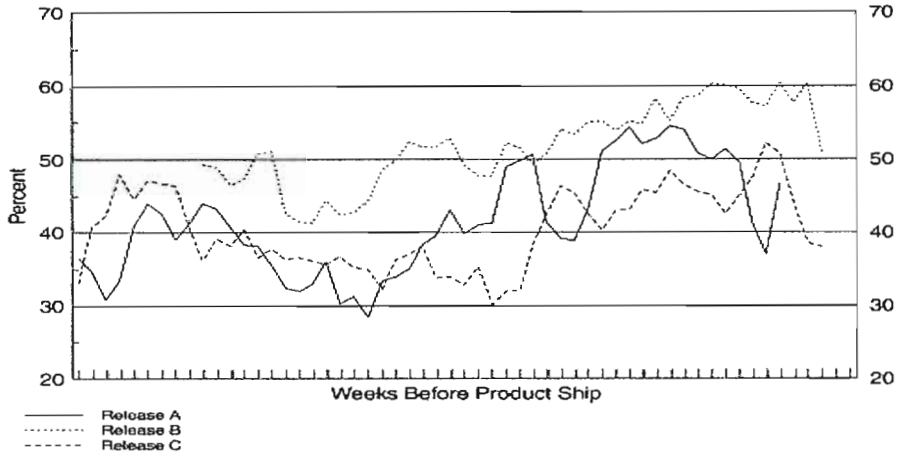

Figure 3.8 Le suivi des pourcentages de défauts de niveaux de sévérité 1 ou 2 dans les différentes mises en production d'un projet [49]

- Suivre les nombres des différents types de défauts détectés au cours du projet [45]. Les défauts trouvés dans les modules réalisés sont catégorisés en plusieurs types selon les causes des défauts, le nombre de défauts dans chaque catégorie sera accumulé et supervisé pendant la construction, ils seront présentés dans un tableau de Pareto lors de l'achèvement de projet. Un tel exemple est présenté dans la figure 3.9 suivante : les C_i dans l'axe des X dénotent les différentes catégories de défaut souvent rencontrés durant la construction comme les erreurs de logique, le contrôle de flux du programme ou la définition des variables, etc. Le tableau de Pareto aide les développeurs à identifier et prévenir le plus tôt possible les erreurs rencontrées fréquemment dans les autres projets similaires.

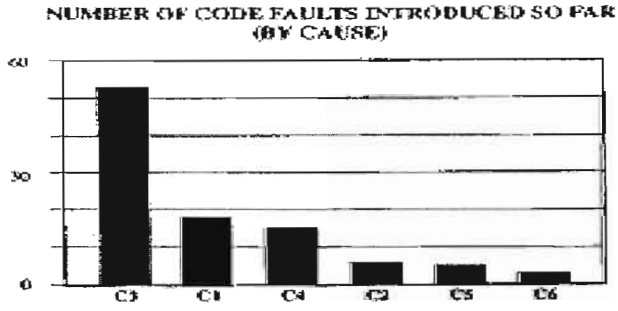

Figure 3.9 Le diagramme de Pareto présentant les différents types de défaut apparus le plus fréquemment dans le code source [45]

La supervision des schémas d'apparitions de défauts permet d'identifier les causes des défauts rencontrés au cours du projet et de proposer des solutions pertinentes pour y remédier

et améliorer la qualité des artefacts réalisés; également, elle nous permet d'anticiper et prévenir les erreurs éventuelles selon l'analyse des schémas référentiels préétablis.

Suivi de la densité de défauts dans le processus d'inspection du logiciel

La métrique de densité de défauts est représentée par le nombre de défauts détectés par unité de volume de code en KLOC ou points de fonction; elle reflète la qualité interne du produit logiciel dans l'activité des tests logiciels [45; 53; 61]. Afin de mieux interpréter les valeurs de cette métrique et d'effectuer des actions correctives dans les délais, la mesure est souvent adaptée en considérant le facteur d'efficacité du processus des tests logiciels qui est une mesure de l'efficacité d'exécution du processus d'inspection des objets réalisés. Elle peut être définie par les différentes formes ci-dessous:

- Dans l'activité de test unitaire du système, l'efficacité du processus de test est représentée par la couverture des jeux de test. Cette dernière est calculée par le pourcentage des chemins totaux exécutés par les jeux de test dans le code source d'un module complet du système;

- Dans l'activité des tests rigoureux du système, elle est introduite par le nombre de jeux de test exécutés par unité de volume de code ou le nombre d'heures dédiées à tester une unité de volume de code par KLOC ou point de fonction.

La figure 3.10 [45] ci-dessous présente un exemple d'utilisation de la mesure de densité de défauts durant les tests rigoureux d'un projet. La densité de défauts est suivie continuellement, l'axe des X représente les valeurs des densités de défauts qui correspondent à chaque activité d'inspection CR_i (Code Review i) pour le projet i : les écarts supérieurs et inférieurs sont indiqués dans le tableau. Les gestionnaires analysent les valeurs obtenues pour prendre les décisions adéquatement en prenant en compte des facteurs comme l'efficacité réelle du processus de tests du système et l'expérience des testeurs. À l'aide de la figure 3.10, nous présentons les différents scénarios sur la densité de défauts [45; 53] :

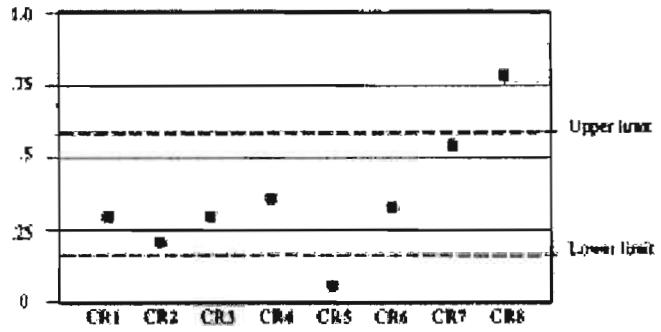

Figure 3.10 Le suivi de la densité de défauts dans les inspections unitaires CR_i d'un projet [45]

- La densité de défaut est inférieure à la limite inférieure comme celle de l'activité CR_5 ; ceci indique que la qualité du code est relativement élevée. Si l'efficacité du processus d'inspection n'est pas conforme à celle qui a été estimée dans la planification, ou si l'inspection de code est faite par des testeurs avec moins d'expérience, les gestionnaires pourront prendre les décisions appropriées comme la demande de réinspection de code, ou également le réajustement des ressources humaines afin d'affecter plus d'effort pour la réinspection.

- La densité de défauts est supérieure à la limite supérieure comme celle de l'activité CR_8 ; si l'inspection est moins efficace que celle prévue, ça veut dire que la valeur réelle de l'efficacité d'inspection est inférieure à la valeur planifiée, ou aussi que les testeurs ont moins d'expérience. Cela révèle que l'objet construit possède une qualité médiocre : on doit réviser le planning et les charges du projet et plus d'efforts seront mis en place pour remédier aux défauts et améliorer la qualité du code.

3.3.4 Mesurer la progression par la méthodologie EVM (*Earned Value Management*)

Comme nous l'avons évoqué au début de la section 3.3, la méthodologie de la valeur acquise est un système de gestion efficace qui permet de poursuivre et contrôler la progression du développement. Plusieurs mesures sont adoptées pour quantifier le coût et l'échéance du développement; elles sont converties et mesurées en unités dollars afin d'analyser la performance du projet de manière cohérente. La figure 3.11 ci-dessous schématise l'utilisation de cette approche ainsi que les métriques principalement adoptées. Dans les paragraphes suivants, nous allons tout d'abord introduire ces mesures en détail, puis nous analyserons cette approche en fonction des points de vue de Lipke [64].

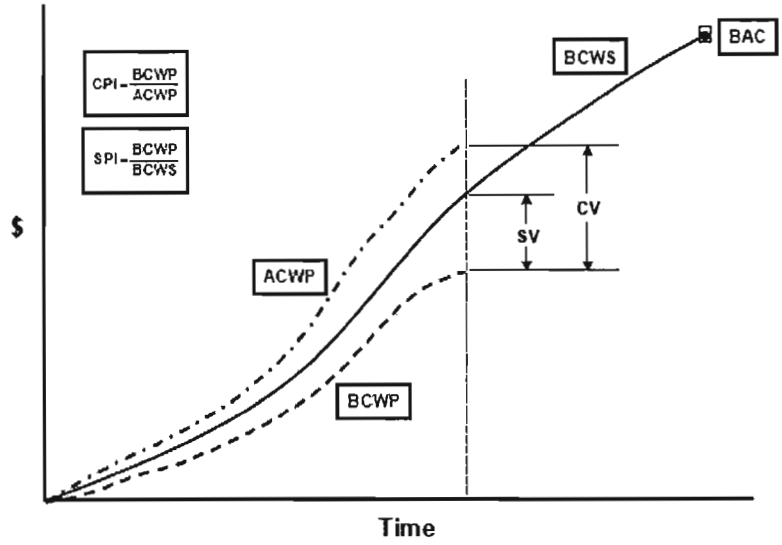

Figure 3.11 Les métriques adoptées dans l'approche EVM par Walt Lipke [64]

Dans la figure 3.11, la progression du développement du projet est mesurée par trois courbes de coûts en forme de « s » [63; 64]:

- La courbe de mesure ACWP (*Actual Cost for Work Peformed*) — la courbe en forme de « s » représente le coût réel du travail effectué à une date donnée;
- La courbe de mesure BCWS (*Budgeted Cost of Work Scheduled*) — la courbe contient les coûts budgétés du travail prévu, ce sont des budgets à date qui sont assignés aux travaux planifiés à la date donnée; le point terminal BAC (Budget At Completion) dans la courbe de BCWS est le coût prévisionnel à la date de complétion de projet;
- La courbe de mesure BCWP (*Budgeted Cost of Work Performed*) — la courbe correspond aux valeurs acquises qui sont des valeurs cumulatives des coûts budgétés du travail complété à la date donnée.

Les indicateurs de progression de développement sont dérivés à partir de ces trois courbes, ils sont classés en deux types :

- Les indicateurs de progression en coûts — la métrique CV (*Cost Variance*) qui indique l'écart de coût entre le coût BCWP et le coût ACWP sous la forme suivante : $BCWP - ACWP$; la métrique d'indice de performance en temps CPI (*Cost Performance Index*) qui est calculée par la division entre le coût BCWP et le coût ACWP : $BCWP / ACWP$;

- Les indicateurs de progression en temps — deux métriques sont impliquées : la métrique SV (*Schedule Variance*) et la métrique SPI (*Schedule Performance Index*). SV est mesurée en unité monétaire (dollars); la métrique SV indique la différence entre la valeur de BCWP et la valeur de BCWS : $BCWP - BCWS$; la métrique SPI est l'indice de performance en temps, elle est calculée par la division de la valeur BCWP par la valeur BCWS : $BCWP / BCWS$;

En analysant les indicateurs en termes de coûts, si la valeur de mesure CPI est supérieure à 1 ou la valeur de mesure CV est supérieure à 0, cela signifie que le projet se déroule en respectant le coût budgétaire; dans le cas contraire, le projet est en surcoût.

En analysant les indicateurs en termes de temps, si la valeur de mesure SV est inférieure à zéro et la valeur de mesure SPI est inférieure à 1, le projet avance moins vite que la planification prévue, le projet est en retard; dans le cas contraire, le projet est en avance.

Analyse de l'approche EVM

Selon Lipke [64], les indicateurs en temps de l'approche ne sont pas aussi fiables que les indicateurs en coûts, les courbes de leur valeurs cumulatives révèlent les anomalies durant le processus de suivi; la problématique est illustrée par les deux figures 3.12 et 3.13 suivantes :

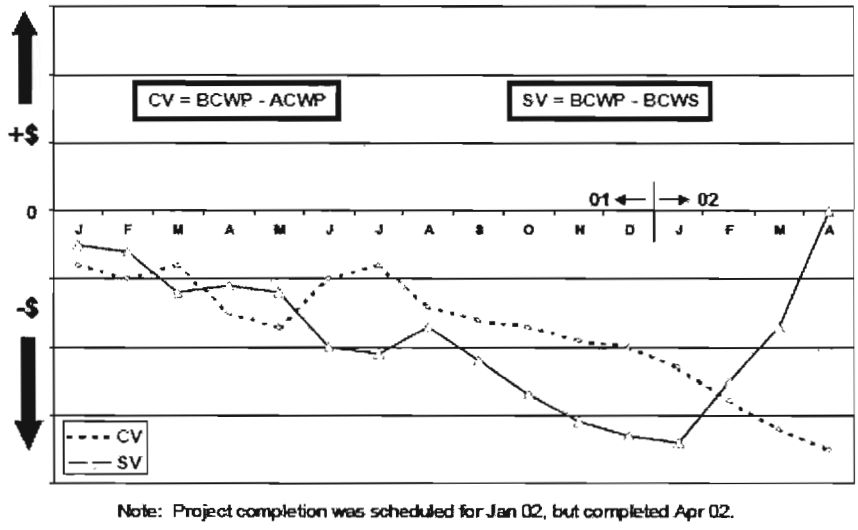

Figure 3.12 Exemple d'utilisation des mesures SV et CV dans un projet [64]

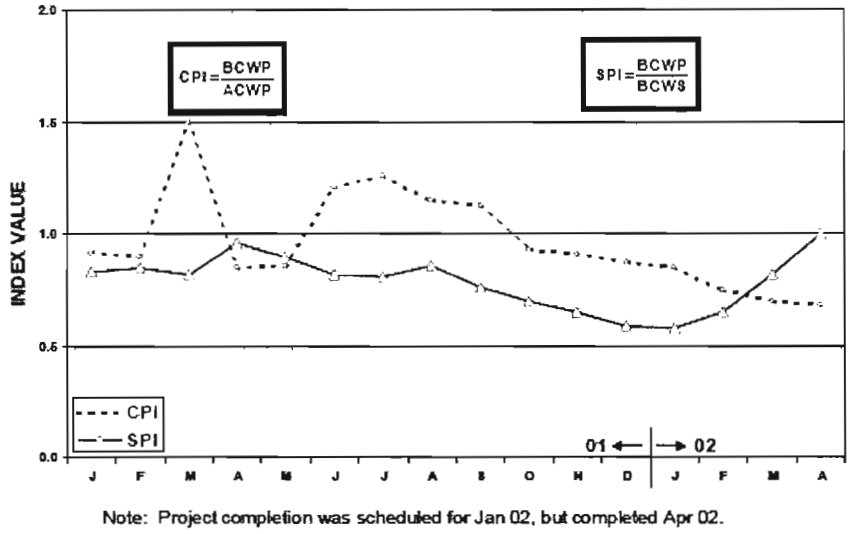

Figure 3.13 Exemple d'utilisation des mesures SPI et CPI dans un projet [64]

La figure 3.12 représente la progression du développement d'un projet par un suivi de la métrique d'écart de coût (CV) et de la métrique d'écart de temps (SV). Comme indiqué dans la figure, le projet est en retard, le problème apparaît dans les valeurs cumulatives qui correspondent à la période allant de la date Janvier 02 jusqu'à la complétion de projet en abscisse; ces valeurs dans la courbe montrent une croissance à pic et convergent vers la valeur zéro à la fin du projet. Cela contredit non seulement le fait du retard de projet, mais présente encore une information peu fiable à l'effet que le projet est achevé dans le temps prévu. De la même façon, la valeur cumulative de la mesure SPI finit à la valeur 1 dans la figure 3.13.

D'après Lipke [64], le problème décrit ci-dessus est dû au fait que les mesures SV et SPI sont obtenues basées sur la mesure BCWS; cette dernière finira vers le point final BAC qui est le coût budgétaire prévu lors de l'achèvement du projet; c'est également la même valeur finale pour la mesure BCWP. Lipke affirme qu'il est inapproprié de représenter la mesure de progression en temps par la métrique de coût de développement, car ça ne reflète pas véritablement le progrès du projet en termes de temps. De plus, les gestionnaires risquent d'avoir des informations trompeuses sur le statut du projet, ce qui rend difficile aux gestionnaires de pouvoir juger si le projet sera complété et livré dans les délais.

Lipke a proposé une nouvelle métrique ES (Earned Schedule) qui mesure la progression en temps de manière plus explicite et plus fiable; la méthode de calcul de la valeur ES et son utilisation sont illustrées dans la figure 3.14 suivante :

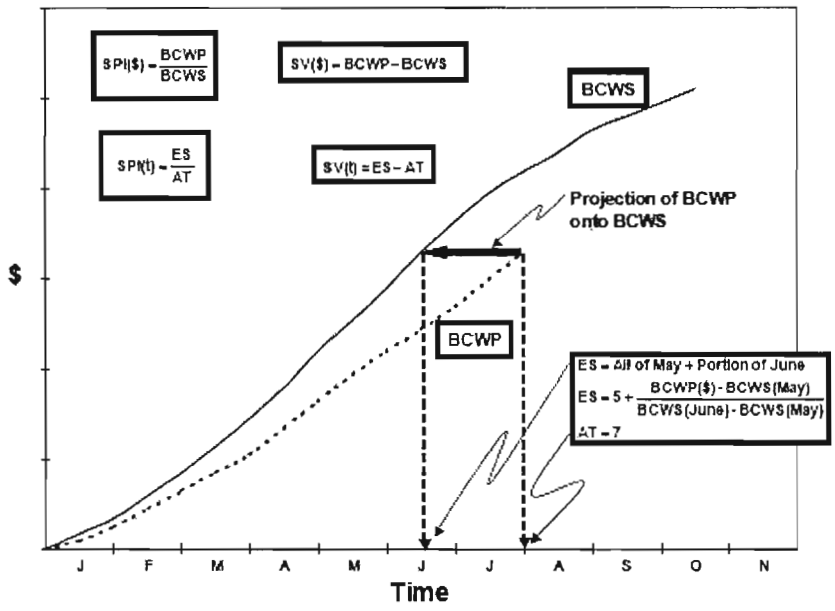

Figure 3.14 Exemple de calcul de la valeur de mesure ES [64]

Comme le montre la figure 3.14, la mesure de ES est calculée par les étapes suivantes :

- Tout d'abord, une projection est faite de la courbe de valeur acquise BCWP vers la courbe de mesure BCWS; la valeur de la mesure ES est identifiée en abscisse par la date associée au point projeté dans la courbe BCWS;

- La valeur de la mesure ES est calculée par l'addition de deux tranches de temps, la première est le temps cumulé jusqu'au début de la période de temps qui se termine à la date associée au point projeté. Cette période est dénotée comme la période incomplète; la deuxième tranche de temps est fractionnaire, elle est calculée par la valeur de BCWP de la période incomplète divisée par la valeur de BCWS planifiée pour la période complète (par ex. le coût budgétaire planifié pour tout le mois de Juin), le calcul est exprimé comme suit :

$\frac{BCWP - BCWS(n-1)}{BCWS(n) - BCWS(n-1)}$ où la valeur BCWP correspond à la date réelle, $BCWS(n)$ dénote la valeur cumulative de la mesure BCWP correspondant à la date n, ainsi que pour $BCWS(n-1)$;

- La figure 3.14 démontre un tel exemple de calcul de la valeur ES; la valeur de la mesure ES est déterminée par le temps cumulé de cinq mois plus la portion du mois de Juin. La dernière valeur est obtenue par la division par la différence de la valeur réelle de BCWP en Juin et de la valeur de BCWS du mois de Mai;

- Une fois la valeur de ES déterminé, la mesure SV est calculée par une soustraction entre la valeur de ES et la valeur de la mesure AT (Actual Time) ce qui signifie le temps cumulé réel. Elle est exprimée par $SV = ES - AT$; le calcul de la mesure de SPI est obtenu par la division de la valeur de ES par la valeur AT : $SPI = ES / AT$;

- Si la valeur de SV est supérieure à 0 ou la valeur de SPI est supérieure à 1, le projet est en avance; dans le cas contraire, le projet est en retard.

Après les validations empiriques faites par Lipke, il conclut que les valeurs de SV et SPI obtenues par l'utilisation de ES mesurent l'avancement du projet de manière plus fiable et plus précise que celles de l'approche de EVM.

3.3.5 Conclusion pour les métriques de progression

À partir de la revue des métriques appliquées dans la mesure de la progression de la construction, nous établissons nos propres points de vue comme suit :

- Du coté positif, ces métriques jouent des rôles multiples dans le domaine de la gestion de projet; l'avantage le plus significatif réside dans le fait que ces métriques fournissent une meilleure visibilité aux gestionnaires sur le statut du développement du projet. Elles sont utilisées afin d'aider les prises de décisions tout en améliorant la qualité et la performance du processus de construction. Cependant, les courbes du schéma référentiel sont établies au début de projet, et servent d'objectifs au projet en termes de coût, de temps et de qualité. Le contrôle du projet sera piloté par la comparaison entre les valeurs réelles et les valeurs de ces objectifs. Par la suite, les gestionnaires pourront effectuer des jugements raisonnables pour ce qui concerne les actions décisionnelles suivantes:

- prendre les actions correctives afin de corriger efficacement les erreurs intervenues;
- prédire et prévenir les risques et les problèmes potentiels dans la construction;
- déterminer si la progression du projet est en avance en termes du temps prévu;
- déterminer si la charge consommée par le projet est en surcoût;
- effectuer le réajustement des ressources de développement pour inspecter et remédier aux défauts apparus, achever les tâches difficiles et rattraper le retard du projet;
- déterminer si la qualité et les fonctions des artefacts livrables sont conformes aux exigences des clients et aux objectifs qualitatifs prédéfinis au début de projet.

• Bien que ces métriques offrent divers objectifs d'utilisation dans le contrôle du projet, il existe encore des inconvénients évidents lors de leur utilisation. Tout d'abord, le suivi et le pilotage de ces mesures sont basés sur les schémas référentiels, ces derniers sont prédéterminés en fonction des mises en production précédentes d'un projet ou de projets similaires. Ils sont inévitablement influencés par les jugements subjectifs des gestionnaires; l'imprécision et la non fiabilité résident toujours dans les courbes des schémas référentiels prédits. Deuxièmement, l'adoption des modèles de développement est un autre facteur essentiel qui affecte le progrès du développement, l'intégrité et la convivialité des mesures de la progression dépendent aussi du modèle de développement adopté. Dans la section suivante, nous allons analyser les métriques en fonction de leur adaptabilité et convivialité dans les différents modèles de développement.

3.4 Les métriques sous les différents modèles de développement

Aujourd'hui, il existe de nombreux modèles de développement qui sont apparus tout au long de l'évolution du génie logiciel. Selon l'histoire des modèles de développement fournie par Larman et Basili [65], le cycle de vie en cascade fut inventé par le département de la défense des États Unis au début des années 70; ce modèle réalise le produit logiciel par une séquence de phases dans un ordre strict : analyse, conception, codage et tests du logiciel. La

conformité de l'atteinte des objectifs de chaque phase doit être validée avant de passer à la phase suivante, mais le modèle ne représente pas la réalité du développement de logiciel qui est un processus très complexe, continu et itératif. Il tend à organiser le développement d'une manière rigide et lourde, car le modèle en cascade ne permet pas de revenir en arrière au cours du projet; le projet avance sans avoir les réactions des clients, ce qui résultera en la méconnaissance des nouvelles exigences des clients. Il est donc très difficile et très coûteux d'adapter les nouveaux besoins des clients dans ce modèle; ces derniers peuvent exiger de refaire toutes le phases du modèle, et on risque également d'avoir le problème de la détection tardive des défauts qui causera un surcoût pour corriger ces défauts.

Des innovations ont vu le jour au milieu des années 80 : le modèle incrémentiel est proposé par Tom Gilb [66]. Ce modèle divise le développement en plusieurs incréments ou sous-systèmes qui sont plus contrôlables en termes d'envergure et de coût de développement. Chaque sous-système sera développé avec un modèle en cascade, cependant, les activités telles que la collaboration interactive avec le client et le réajustement du planning du projet doivent être mises en place vers la fin du cycle de vie de chaque incrément. En 1988, Boehm a introduit le modèle de développement en spirale [68]. Cette démarche tente de piloter et perfectionner le processus de développement par l'évaluation des risques potentiels, et permet d'assurer la conformité aux objectifs qualitatifs dans chaque incrément de développement. Les documents d'évaluation du risque servent de directives significatives afin d'élaborer la conception du système tout en évitant les risques éventuels et assurant la qualité du produit final.

Plus récemment, les modèles Agile comme la méthode XP (Extreme Programming), la méthode Scrum, la méthode FDD (Feature-Driven Development) sont mis au point afin de rendre le développement plus efficace [67]. Ces méthodes mettent l'emphase sur les activités telles la communication entre les individus au sein de l'équipe de développement, l'interaction très collaborative avec les clients, l'intégration et les tests rigoureux continus lors du développement. Ces modèles présentent des avantages par leur souplesse et leur facilité d'adaptation aux changements au cours du projet et l'assurance de qualité du produit livrable qui satisfait les exigences des clients.

Dans les sous-sections suivantes, on s'attarde sur les trois types de modèles de développement les plus classiques:

- Le modèle incrémentiel
- Le modèle de développement en spirale
- Les modèles Agile
    - La méthode XP
    - La méthode Scrum
    - La méthode DSDM (Dynamic System Development Method)

D'abord, on va détailler les fondements de chaque modèle de développement, puis on fait une comparaison horizontale en termes de leur influence dans la gestion de projet en fonction de caractéristiques propres aux différents modèles. Finalement, on propose les méthodes et les mesures qui sont adoptées avec chaque modèle de développement.

3.4.1 Le modèle incrémentiel

D'après le point de vue de Martin [69], les difficultés liées au modèle en cascade sont listées ci-dessous dans un contexte de gestion de projet :

- Le développement du modèle en cascade s'exécute séquentiellement, le retour à la phase précédente est limité, et il n'y a pas vraiment de réactions des clients entre les phases du modèle. La phase de réalisation du système est strictement basée sur les documents de conception et d'analyse; ces derniers devront être réajustés et raffinés graduellement lors du déroulement de l'activité de réalisation; il est donc difficile de maîtriser la réalisation d'un système complexe avec ce modèle.

- En pratique, il est inapproprié de placer les jalons à la fin de chaque phase du modèle en cascade pour refléter la progression du développement. Ceci est dû au fait que le développement est un processus complexe, et que l'analyse et la conception seront fréquemment modifiées selon les réactions des clients. Ainsi, les améliorations du code, et le processus de développement vont progresser par améliorations continues. Avec le modèle en cascade, il est inadéquat et imprécis de déterminer la complétion des phases en fonction des échéances prédéfinies; également, le modèle ne permet pas la participation des clients au cours du développement, et les analystes risquent d'avoir une mauvaise compréhension des besoins, ce qui résultera en prises de

décision trompeuses et éventuellement en la reprise de toutes les phases; le projet s'achèvera donc trop tardivement.

Afin d'éliminer ces contraintes du modèle, selon les expériences de Martin [69] et May et Zimmer [66], le modèle incrémentiel est établi en respectant les caractéristiques constatées ci-dessous pour mieux contrôler le développement d'un projet:

- Le système est décomposé en plusieurs incréments, chaque incrément produit un objet exécutable et livrable au client dans un période courte (1 à 4 semaines), l'incrément est réalisé par une production dans laquelle on adapte le cycle de vie en cascade qui correspond à un développement relativement simple;

- À chaque mise en production d'incrément, les développeurs mettent à jour des nouvelles fonctions afin de répondre aux nouvelles exigences des clients. Ensuite les intégrations interviennent progressivement à la fin de chaque incrément;

- La participation collaborative des clients doit être ajoutée à la fin de chaque mise en production sur l'artefact partiel construit par l'incrément. C'est afin d'avoir leur réaction tout en comblant les fonctions du système de manière itérative, et également en fonction des expériences provenant des incréments successifs, le réajustement du planning du projet et l'estimation du prochain incrément sont effectués en termes de sa durée, de son coût et de son envergure.

La figure 3.15 ci-dessous illustre l'utilisation d'un tel modèle de développement en comparaison avec le modèle en cascade :

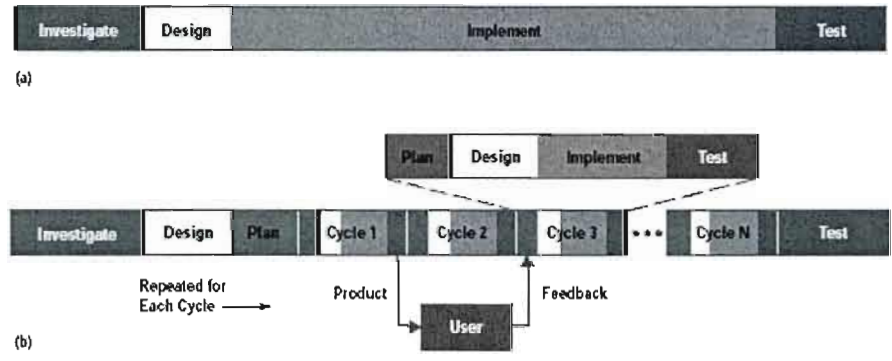

Figure 3.15 a) Le modèle de développement en cascade b) Le modèle de développement incrémentiel donné par May et Zimmer de la compagnie Hewlett-Packard [66]

À partir des revues [66; 69] et également d'expériences industrielles dans la compagnie informatique FIRM qui effectuent une substitution d'utilisation du modèle en cascade par

l'adoption du modèle par incrément [70], nous établissons le tableau de comparaison 3.1 suivant afin de démonter les bénéfices et les impacts positifs du modèle dans un perspective de gestion de développement :

Modèle en cascade (Inconvénients)	Modèle incrémentiel (Avantages)
• Analyse des besoins mettant l'accent sur les fonctions du produit mais les aspects de qualité du produit ne sont pas considérés; • Sans réactions du client, les analystes comprennent souvent mal les besoins; • Les exigences sont stipulées et stabilisées très tôt dans le développement; il est difficile de s'adapter aux nouvelles exigences durant la réalisation; • Manque de réaction des clients sur la convivialité et la performance de l'objet construit; donc une méconnaissance de savoir si le produit se conforme aux objectifs qualitatifs; • Détection et correction tardives des problèmes vers la fin du projet, ceci causera des surcoûts pour le redéveloppement afin de corriger le tir;	• Le système est décomposé en morceaux qui sont plus faciles à gérer, plus simple à réaliser; • Les objectifs de qualité de chaque de mise en production du système sont exprimés de manière précise et explicite; • Également la planification de la réalisation d'un incrément sera faite en termes de temps et de coûts selon les expériences obtenues des mises en production terminées; • Le test fonctionnel et l'intégration à la fin de chaque mise en production permet d'identifier les problèmes survenus le plus tôt possible et de les réparer à temps; • Après chaque mise en production, les réactions des clients en termes de convivialité et performance du produit partiel sont directement livrées aux développeurs selon les utilisations par les clients; l'amélioration de la qualité du code sera conduite par la collaboration des clients; • Augmentation de la motivation au sein de l'équipe de développement provoquée par la communication fréquente entre les clients et les développeurs

Tableau 3.1 La comparaison entre le modèle en cascade et le modèle incrémentiel

Tel qu'énoncé ci-dessus, on peut affirmer que le modèle incrémentiel rend le développement plus contrôlable et ainsi plus souple en adoptant les changements au cours du projet. La

qualité du système est raffinée et améliorée de manière évolutive; on devra établir les objectifs qualitatifs à la fin de chaque mise en production du système, ces objectifs sont considérés être les indicateurs de l'évolution de la qualité de l'artefact partiel de chaque mise en production. Il sera significatif d'adopter des mesures de qualité comme le temps moyen d'échec du fonctionnement de l'artefact partiel et exécutable et la densité de défauts afin de révéler la fiabilité de l'artefact partiel. Le pourcentage des clients satisfaits ou non satisfaits représente la convivialité et la facilité de maintenance du système partiel; ces mesures fournissent plus de visibilité au statut réel du projet tout en aidant aux prises de décision des gestionnaires et tentent de piloter la maîtrise de projet afin de satisfaire aux exigences des clients.

3.4.2 Le modèle de développement en spirale

Le modèle en spirale fut initialement proposé par Boehm en 1988; il vise à améliorer le processus tout en réduisant les risques, particulièrement pour un système gros et complexe. Le modèle est centré sur l'évaluation des risques dans les phases préliminaires de manière cyclique. L'évaluation porte sur une liste des risques encourus comprenant des sujets tels que les enjeux personnels, la planification irréaliste en termes de temps prévu et de budget, les changements continus des besoins des clients, le développement des fonctions inadéquates et les problèmes de performance, etc. Un tel modèle de développement est illustré par la figure 3.16, particulièrement pour le développement d'un système logiciel gros et complexe.

Selon cette figure, on constate que chaque cycle du modèle comporte quatre étapes:

- Identification des objectifs qualitatifs de l'artefact du cycle de développement en ce qui concerne la fiabilité, la performance et les fonctions. Détermination des alternatives de développement en considérant les contraintes imposées dans ces alternatives en termes de coût et de calendrier;

- Identification et évaluation des risques potentiels dans la construction éventuelle. Cependant, les solutions aux risques doivent être proposées et vérifiées via les activités comme la révision du planning du projet, le développement du prototype, la communication entre les clients et les développeurs. Le prototype est une application exécutable permettant d'évaluer les risques quantitativement et de valider les solutions retenues expérimentalement; les risques seront éliminés par l'utilisation d'une série de prototypes évolutifs;

- Une fois les risques réduits à un niveau acceptable, le cycle du projet sera construit en suivant le modèle en cascade ou le modèle incrémentiel;
- Enfin, on doit vérifier l'artefact produit par le cycle et faire la planification pour le prochain cycle de développement.

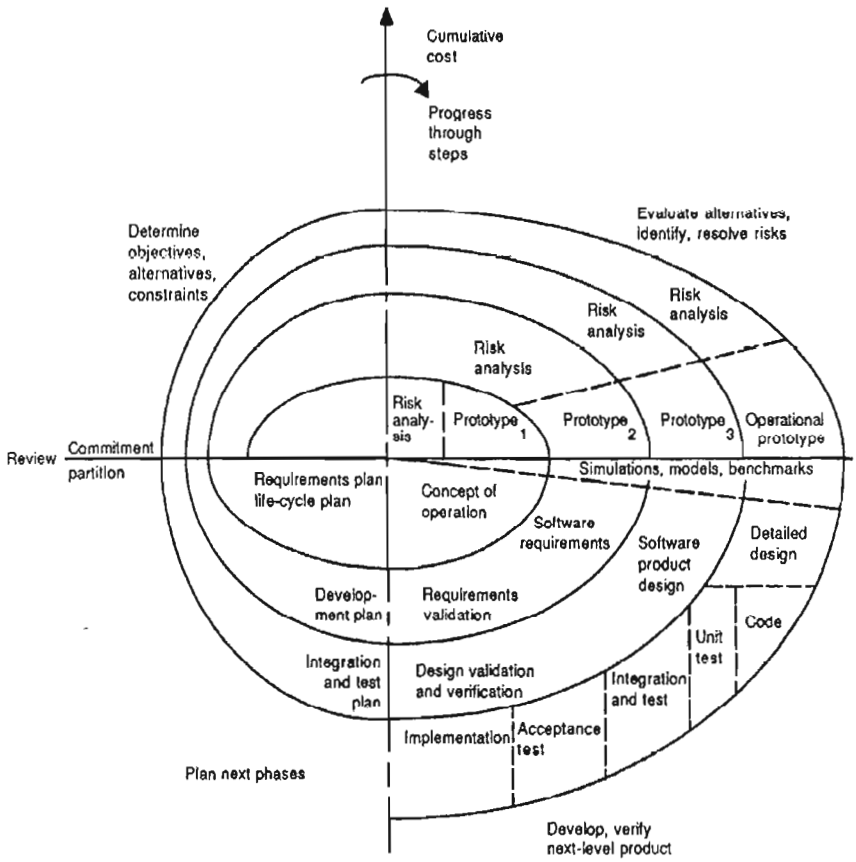

Figure 3.16 Le modèle en Spirale donné par Bohem [68]

En fonction des revues [66; 68; 69; 71], nous établissons le tableau 3.2 ci-dessous afin de comparer les caractéristiques entre les trois modèles de développement classiques : le modèle en Spirale, le modèle incrémentiel et le modèle en cascade. Cette table a été établie pour un projet de grande envergure, instable, complexe, souvent modifié à cause de nouvelles exigences du client et exigeant des intégrations fréquentes des nouveaux modules; pour ce type de projet, on sait déjà que le modèle en spirale est plus fiable et plus performant que les

deux autres modèles. On ne considère donc que les avantages du modèle en spirale et les inconvénients des deux autres.

Le modèle en spirale (Avantages)	Le modèle incrémentiel (Inconvénients)	Le modèle en cascade (Inconvénients)
• Les exigences des clients sont découvertes et précisées de façon continue; • Le développement est piloté par l'évaluation des risques, cependant, l'utilisation de prototypes fournit la souplesse pour adopter de nouvelles exigences des clients et valider les solutions proposées pour des risques identifiés; • L'architecture de conception sera évaluée et complétée graduellement entre les cycles d'analyse de risques en utilisant les prototypes qui évaluent les risques quantitativement; • La planification du projet est ajustée et raffinée itérativement lorsqu'une solution plus fiable aux risques identifiés ou une meilleure compréhension des exigences des clients sont retenues.	• Dans chaque incrément du modèle, la mise en oeuvre de nouvelles exigences des clients affecteront les incréments précédents construits dans lesquels l'architecture du programme est déjà conçue et stabilisée, ce qui sera difficile à modifier; • Le modèle permet de développer les incréments parallèlement, et la difficulté réside dans l'intégration de chaque incrément au système global.	• Les exigences des clients sont spécifiées et stabilisées très tôt dans le développement et il est difficile d'adopter les changements; • Il n'y a pas vraiment d'évaluation du risque pertinente pour les exigences obtenues, la détection des défauts ou des enjeux graves sera tardive; • L'architecture est élaborée et stabilisée selon une analyse des besoins incomplète et imprécise; la remise en cause de l'architecture du système sera difficile; • La planification n'est pas assez précise en ne considérant pas les risques possibles durant le développement; également elle est souvent basée sur le temps prévu ce qui est inapproprié pour refléter la progression.

Tableau 3.2 Tableau de comparaison entre les trois modèles de développement

Basé sur les propriétés des modèles listés ci-dessus, Brownsword et Smith [71] apportent des modifications aux mesures de la méthode EVM (*Earned Value Management*) qui est souvent adoptée dans le modèle en cascade. Ceci a pour but de rendre les mesures de EVM plus pertinentes et plus précises tout en mesurant la progression avec le modèle en spirale; on démontre l'utilisation de ces mesures modifiées et la détermination de leur valeurs dans les étapes suivantes :

- Lorsque le projet est planifié, les tâches sont identifiées et ordonnées dans une structure hiérarchique; on devra établir la valeur de BCWS (*Budget Cost Work Schedule*) qui représente le coût budgété associé à chaque tâche. La valeur de BCWS est calculée de la façon suivante :

$$\left|BCWS_i\right| = (BCWS_i * eRisk_i) * \sum_{j=1}^{n} BCWS_j \Big/ \sum_{k=1}^{n} (BCWS_k * eRisk_k)$$

où le coût planifié de chaque tâche sera pondéré par un coefficient de risque, dénoté par $eRisk_i$; il indique la criticité de la tâche i dans le développement. Le terme $\sum_{j=1}^{n} BCWS_j$ dénote la somme des coûts pour toutes les tâches, le terme $\sum_{k=1}^{n} (BCWS_k * eRisk_k)$ représente la somme des coûts des tâches qui sont pondérés par la multiplication par le coefficient de risque;

- D'après Brownsword et Smith, la valeur acquise BCWP (*Budget Cost Work Performed*) dans le modèle en spirale doit refléter la caractéristique essentielle du modèle en ce qui concerne une réduction cumulative des risques subsistants à travers le temps. La valeur acquise de chaque tâche dans le temps T est formée comme suit :

$$EV_T = \left|BCWS_i\right| \times \left(\frac{1 - rRisk_T}{1}\right), rRisk_T \in [1 \quad \delta]$$

où la notion EV (*Earned Value*) est la valeur acquise calculée, le terme BCWS est le coût planifié de chaque tâche, le terme $rRisk_T$ est le pourcentage de risques subsistants. Il va de 1 jusqu'à la valeur δ, ce qui représente un seuil acceptable lors de l'achèvement de la tâche; la valeur de $rRisk_T$ est déterminée

par les gestionnaires selon la situation réelle de l'environnement de développement;

- Enfin, le coût budgété du projet global dans le temps T, $BCWS_T$ est exprimé comme suit :

$$BCWS_T = \sum_{j=0}^{T} \sum_{i=1}^{n} EV_{ij}$$

La formule représente la somme des valeurs acquises de toutes les tâches depuis le début de projet jusqu'au temps T.

3.4.3 Le modèle Agile

Les méthodes Agiles sont conçues pour favoriser l'amélioration du processus de développement de manière itérative; le projet est développé dans un cycle de vie simple et court en appliquant une discipline pragmatique afin d'accélérer le développement et d'achever le projet plus efficacement. Ces méthodes permettent de faire évoluer et de raffiner constamment la planification, en encouragent la collaboration du client et en effectuant les tests fonctionnels du logiciel tout au long du projet; ceci donne de l'Agilité au projet pour pouvoir répondre aux changements. Dans ce qui suit, nous faisons une description brève des trois méthodes principales de l'approche Agile [67]: la méthode XP (*Extreme Programming*), la méthode Scrum et la méthode DSDP (*Dynamic System Developement Method*). Par la suite, nous décrivons les caractéristiques communes de ces méthodes en énumérant leurs propres impacts positifs et négatifs dans un contexte de gestion de projet. Enfin, nous exposons et analysons les métriques de progression adoptées par les méthodes Agile.

La méthode XP

La mise en oeuvre de la méthode consiste en quatre phases respectives [76]: *la phase d'exploration* dans laquelle les exigences sont préparées et fournies par les clients durant un mois. En même temps, l'équipe de développement se familiarise avec les pratiques, l'environnement, ainsi que l'outil de développement. *La phase de planification* consommera quelques jours pour compléter l'estimation d'effort et l'établissement du calendrier pour la première mise en production du système. *La phase d'itération* effectuera plusieurs itérations pour produire la première production; chaque itération durera un mois au maximum et le test

fonctionnel est effectué à la fin de chaque itération. *La phase de mise en production* comprend des activités comme les tests rigoureux afin d'assurer la qualité et la performance des livrables, et la prise de décision pour adapter les changements intervenus dans la mise en production. Pendant le déroulement du projet, un ensemble de critères pratiques devra être respecté afin d'offrir rapidité et fiabilité à la mise en oeuvre du projet; il s'agit des principes suivants :

- la participation du client au planning pour déterminer les objectifs du projet et évaluer les risques potentiels avec l'équipe de développement;
- les mises en production sont développées rapidement dans des cycles courts et devront être livrées fréquemment aux clients afin d'avoir des réactions le plus tôt possible;
- les activités telles la construction et le test continu du code seront effectuées en binôme pour assurer la qualité du code et éliminer le risque de retard du projet en cas d'absence d'un programmeur;
- l'intégration continue des modules construits et testés dans le système; ceci permet de détecter les erreurs plus tôt dans le développement et réduit l'effort de correction.

La méthode Scrum

C'est une méthode de gestion simple et performante se concentrant sur la façon d'organiser l'équipe afin d'adopter les changements au cours du projet; c'est la méthode la plus souvent utilisée pour la gestion de projet Agile. La méthode possède les caractéristiques suivantes [76] :

- Le planning du projet est basé sur une liste réservée concernant les exigences des clients; la liste sera souvent mise à jour lorsqu'il y a des changements. La conception du système est également conçue et raffinée en fonction de la liste des exigences; avec ce modèle, les changements sont identifiés très tôt durant le développement;
- Le développement se fait par des itérations de durée courte d'un mois au maximum; les itérations sont appelées « Sprints ». Chaque « Sprint » a pour but de développer un objectif du projet de la liste de fonctions réservées. À la fin de chaque « Sprint », le test fonctionnel sera appliqué; les objectifs du projet sont analysés et calibrés quotidiennement par les réunions des gestionnaires.

La méthode DSDM (Dynamic System Development Method)

Cette méthode fournit un cadre de gestion de développement efficace. En premier lieu, la méthode tente de planifier le projet en considérant les contraintes imposées en termes d'échéances et de ressources de développement disponibles. Ensuite, on ajuste le nombre des fonctions à développer afin d'assurer le succès du projet. Le processus de développement DSDM suit cinq phases principales [76]:

- La phase d'analyse de faisabilité du projet, où l'utilisation de la méthode DSDM sera évaluée en considérant des facteurs tels que le type du projet, les facteurs humains, la technologie employée, etc. La phase génère le rapport de faisabilité et le document des grandes lignes du projet;

- La phase d'étude des affaires qui sert à spécifier les fonctions du système selon les exigences des clients. Dans cette phase, on détermine l'architecture initiale du système et le plan du prototype en définissant la stratégie du prototype ;

- Dans la phase itération du modèle fonctionnel, les spécifications de fonction du projet seront améliorées et priorisées par l'utilisation des prototypes de façon itérative; les risques éventuels du développement sont aussi évalués;

- La phase de conception et de réalisation itérative produit un artefact livrable qui satisfait aux minimum d'exigences des clients. Ensuite, l'artefact sera raffiné et complété selon les réactions des clients à plusieurs itérations de développement. Cependant, les artefacts intermédiaires doivent être livrés fréquemment aux clients pour évaluation;

- Dans la phase de mise en oeuvre, le produit final est livré au client qui sera formé pour pouvoir utiliser le produit logiciel et prendre en charge l'entretien du système construit.

Coram et Bohner [67] ont constaté les spécificités partagées entre les différentes méthodes Agiles. Le tableau 3.3 ci-dessous montre les fondements de l'approche Agile en en citant les valeurs fondamentales ainsi que les avantages et les inconvénients pour la gestion de projet:

Caractéristiques Agile	Impacts positifs	Impacts négatifs
• Privilégier la communication au sein de l'équipe de développement et l'interaction au maximum entre le client et l'équipe; • Favoriser l'implémentation et l'inspection de code tout au long de la construction; • Les équipes de développement sont de petite envergure : de 10 à 15 développeurs; • Minimaliser et simplifier le cycle de développement et les livraisons de productions fréquentes; • Malgré la planification de projet dynamique, les échéances de chaque mise en production de système sont figées et imposées. Mais les fonctions à mettre en oeuvre à chaque mise en production sont souples et variables au cours du projet;	• Encourage la collaboration dans l'équipe de développement en permettant d'augmenter la productivité de l'équipe et l'efficacité du processus de développement; • L'implication du client durant le processus de développement réduit le risque de méconnaissance des exigences des clients; • Rendre plus de liberté et d'autonomie aux développeurs afin de réaliser le logiciel; • Rapidité et efficacité pour adapter les exigences réelles aux opinions des clients de manière pertinente. Ceci réduit l'effort de développement des fonctions non exigées pour le cycle actuel; • Via la construction et l'inspection de code en binôme au cours du processus, on pourra améliorer la qualité du code réalisé. Ceci élimine aussi le retard de développement en cas d'absence d'un développeur clef;	• La petite équipe de développement met une exigence forte sur la compétence et l'expérience des développeurs; • Les développeurs ne sont pas familiers avec la programmation en binôme; • Le principe de minimalisation du cycle de construction ne développe que les fonctions exigées sans considérer les exigences éventuelles, ceci provoque parfois des efforts excessifs pour adopter de nouvelles fonctions; • Mal adapté aux projets de grande envergure qui exigent une longe durée et plus de fonctions à réaliser, car les risques sont parfois inévitables tels la perte de connaissances importantes en cas de démission des membres clefs de l'équipe; • Les contraintes d'obligation contractuelle qui possèdent l'exigence de spécifier et prioriser les besoins des clients dans la planification de chaque mise en production du projet;

• Centré sur l'intégration de nouveaux modules et un test rigoureux du système quotidiennement.	• Le test rigoureux quotidien permet de détecter et de corriger les erreurs le plus tôt possible, tout en réduisant les risques de surcoût pour corriger les défauts plus tard dans le projet; • Construire un produit final assez robuste qui est systématiquement intégré, testé et fonctionnel; • Moins de documentation en réduisant l'effort gaspillé à la production de documentation non nécessaire.	• Moins de documentation signifie plus de difficulté de maintenance du système. Mais le problème de démission des membres de l'équipe est aussi un risque crucial dans la maintenance du produit logiciel.

Tableau 3.3 Les caractéristiques de gestion d'un projet Agile

3.4.4 Mesures utilisées dans le développement Agile

Selon les caractéristiques du tableau 3.3 ci-dessus, les mesures de la gestion de projet Agile devront refléter les perfectionnements successifs des itérations de développement en termes de bénéfice commercial de l'organisation, de qualité du logiciel à chaque mise en production et d'efficacité de l'équipe de développement. Ces mesures pilotent le contrôle du développement Agile, leur utilité principale est composée des choses suivantes [72; 75; 79]:

- Estimer et planifier les fonctions à réaliser dans chaque itération du processus; chaque fonction sera priorisée et ordonnée selon son propre profit commercial estimé lors de la livraison au client;

- Mesurer la progression par les profits commerciaux générés successivement par des itérations; les prises de décision des gestionnaires seront effectuées basées sur ces valeurs clefs afin de piloter le développement vers le succès en termes de commerce;

- Identifier et aviser les développeurs des risques potentiels dus à l'adaptation au changement et diminuant l'efficacité de l'équipe de développement dans chaque itération.

Tel que rapporté par Hartmann et Dymond [72], deux types de métriques sont impliqués dans la gestion de projet Agile :

- La mesure clef par laquelle la maîtrise de projet est pilotée; une telle mesure vise le profit commercial livré par chaque itération du projet. Hartmann et Dymond affirment qu'il est plus approprié et plus souple de quantifier et de valider l'amélioration de processus en termes de valeur commerciale, parce que la progression du modèle en cascade est mesurée par pilotage à partir d'un planning de projet en termes de coûts et d'échéances. Cependant, les exigences des clients sont figées en avance, le développement n'a pas de souplesse et d'agilité afin de répondre aux exigences instables. Les métriques centrées sur le coût et le calendrier ne représentent pas les faits véritables de la progression du projet et fournissent parfois des informations trompeuses aux gestionnaires;

- Les mesures auxiliaires, qui sont appliquées au cours du développement de chaque itération, aident les développeurs à évaluer la qualité du code et l'efficacité du développement tout en améliorant le processus. Elles comprennent des métriques telles que la complexité cyclomatique d'un artefact, le niveau de dépendance entre des modules, la rapidité de développement et le nombre de fonctions testées, etc.

Mesure primaire et clef

Plus récemment, Rawsthorne [78] a modélisé la méthode de la valeur commerciale acquise nommée EBV (*Earned Business Value*). Elle représente le bénéfice réel généré par les mises en production livrées aux clients; cette méthode mesure la progression du projet Agile par les étapes suivantes :

- Au départ, le projet est organisé selon trois aspects principaux : la production, l'équipe et le marketing. Les tâches dans chaque aspect sont identifiées et structurées dans une hiérarchie descendante; cette dernière est illustrée par la figure 3.17. Les sous tâches appartenant aux mêmes tâches sont pondérées par des valeurs relatives; ces valeurs représentent les ratios proportionnels entre ces tâches en considérant leurs propres bénéfices commerciaux lors de l'achèvement du projet. Par exemple, la tâche notée « Feature » apportera 100% de bénéfices commerciaux par rapport à la tâche notée « Structure»; chaque tâche constitue un ensemble de sous-tâches qui réaliseront les fonctions du système;

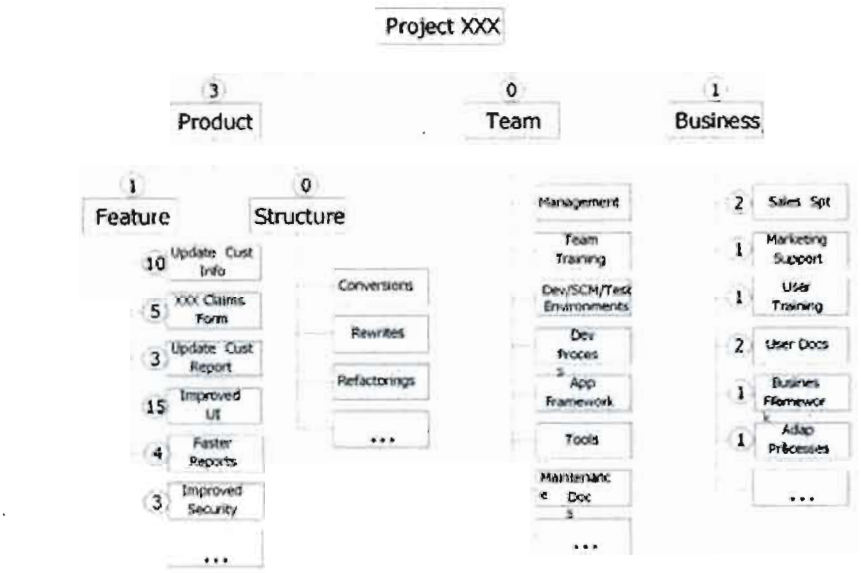

Figure 3.17 Exemple de structure de l'organigramme d'un projet Agile [78]

- De cette façon, chaque fonction du système sera décomposée en une série de mises en oeuvres nommées « *Stories* »; ces dernières représentent les activités unitaires afin de réaliser la fonction. Également, elles sont associées aux poids estimés; la figure 3.18 montre l'exemple d'un ensemble de « *Stories* » qui tente de réaliser la fonction « Mise à jour des informations des clients » du système.

wt	Story
0	Determine what the Stakeholders Want
10	Develop the Main Execution Flow
0	Determine alternative flows
3	Develop Alternative Flow 1
2	Develop Alternative Flow 2
5	Stress-test the feature for 500 simultaneous users

Figure 3.18 Exemple des Stories et de leurs poids pondérés

- Ensuite, on effectue l'estimation du bénéfice commercial (BV — *Business Value*) de toutes les tâches dans l'organigramme; l'estimation se fait de manière récursive en traversant toute la structure de l'organigramme. Le calcul est décrit par la formule générale qui suit :

$$BV(T) = BV(PT) \times \left[weight(T) \Big/ weight(T) + \sum_{Ts'} weight(T') \right]$$

où la notion $BV(T)$ dénote le bénéfice calculé pour la tâche T, le terme $BV(PT)$ est le bénéfice estimé de la tâche parente de cette tâche, la formule $\left[weight(T) \Big/ weight(T) + \sum_{Ts'} weight(T') \right]$ représente le pourcentage proportionnel en termes de bénéfice apporté entre la tâche T et toutes les tâches $(T + Ts')$ qui appartiennent à la même tâche parente, la notion $weight(T)$ est le poids associé à la tâche i;

- La valeur commerciale acquise est obtenue par l'accumulation des valeurs commerciales des tâches (*Stories*) lors de leur accomplissement; le calcul est fait de la façon suivante :

$$EBV(\text{Pr}oject) = \sum_{Completed} BV(Story)$$

où $\sum_{Completed} BV(Story)$ introduit la somme des valeurs commerciales des tâches (*Stories*) complétées.

En utilisant les mesures de bénéfice commercial (*Business Value*) et de bénéfice commercial acquis (*Earned Business Value*), on a les avantages significatives suivants :

- La planification du projet sera souple pour adapter le réajustement des ressources de développement lors d'exigences de nouvelles fonctions; la valeur EBV fournit la meilleure compréhension du progrès actuel de la construction;
- Piloter les prises de décision : les bénéfices commerciaux acquis représenteront la tendance des retours d'investissement. Si une diminution continuelle des profits est découverte, cela révèle que les bénéfices apportés par de nouvelles fonctions en construction ne valent pas leurs propres coûts. On devra stabiliser la conception de fonctions du système et livrer la production au client.

Mesures auxiliaires

Plusieurs métriques auxiliaires sont utilisées de manière coopérative avec la mesure clef dans la gestion de projet Agile [72; 74; 77]. Elles aident les gestionnaires en permettant de détecter et de résoudre les problèmes intervenus en termes de performance de processus et de

qualité d'artefact au bon moment, et aussi de prévenir les risques potentiels qui inhibent l'implémentation des fonctions exigées durant chaque itération. Nous présentons ces métriques dans ce qui suit.

La rapidité de l'équipe de développement [72] — cette mesure est définie comme le nombre de tâches (*Stories*) complétées ou le nombre d'heures idéales de travail dépensées par itération. Divers facteurs influencent la valeur de mesures comme le changement des membres de l'équipe de développement, les compétences des développeurs et la difficulté de la tâche d'implémentation de fonction, etc. La diminution des valeurs de vitesse des itérations peut révéler les problèmes survenus au sein de l'équipe;

La mesure RTF (*Running Test Feature*) [74] — la métrique est représentée par le nombre total des fonctions intégrées et testées dans le projet; les valeurs de mesure doivent s'intensifier linéairement lors du déroulement du projet; ces valeurs cumulatives seront supervisées durant toutes les itérations afin de détecter les problèmes survenus. La figure 3.19 illustre une situation inattendue dans un projet Agile, on voit qu'il y a une descente à pic et une tendance de stabilisation sur la courbe des valeurs cumulatives, cela indique qu'il y a possiblement des changements dans les membres de l'équipe, des modifications dans les exigences des clients, la détérioration de qualité d'artefact, etc. En conséquence, les gestionnaires seront avisés par ces indications et tenteront d'identifier les causes ou les problèmes;

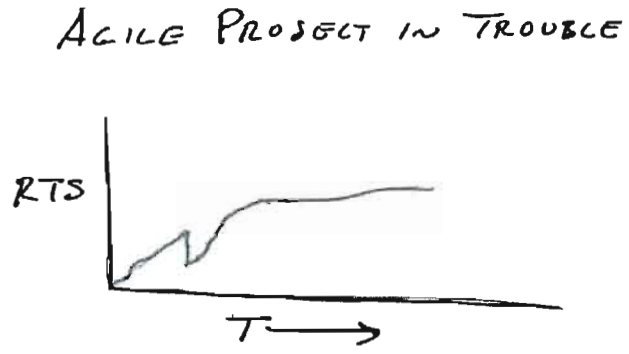

Figure 3.19 La supervision de mesure RTF illustrée par Ron Jeffries [74]

Les métriques de qualité logicielle [77] — selon Knoernschild, l'adaptation des modifications dans les projets Agile provoque souvent l'augmentation des défauts ou des enjeux dans le code construit, plus de difficultés pour la maintenance et le réusinage d'artefact de l'itération. L'utilisation de métriques qualitatives nous permet d'évaluer l'architecture et la qualité d'artefact réalisé tout en identifiant les modules qui exigent d'être calibrés et améliorés. Les mesures qualitatives sont des réactions fiables pour identifier les problèmes qualitatifs du code et faciliter l'adoption des changements ou des modifications. Les métriques proposées sont exposées dans ce qui suit.

- *La complexité cyclomatique* (présentée dans la section 2.1.4 du chapitre 2) est une mesure appropriée qui permet d'évaluer les complexités des modules construits. Également, elle possède une corrélation étroite avec le taux de défauts dans le code. Des modules réalisés avec plus de complexité cyclomatique correspondent à plus de défauts et moins de compréhensibilité dans le code; des réusinages doivent être faits afin de rendre le code plus concis avec moins de défauts, ce qui facilite l'adaptation des modifications dans les itérations successives;

- *Le couplage* entre des modules ou classes; la mesure tente d'analyser les dépendances parmi des modules individuels via deux métriques principales : *le couplage afférent* C_a d'un module qui représente le nombre d'appels reçus des autres modules et *le couplage efférent* C_e d'un module qui signifie le nombre d'appels émis vers les autres modules. Ces deux métriques permettent de mesurer l'intégrité globale de la conception du système tout en évaluant le coût dédié aux changements et la possibilité de réutilisation des modules réalisés. D'une part, un module ayant un couplage afférent excessif sera très coûteux et aura plus de risques lors de l'adaptation de modifications, parce qu'on doit considérer les autres modules qui font des appels au module modifié. Ceci exige de mettre plus d'effort dans les tests d'intégration des modules. D'autre part, un module avec trop de couplage efférent sera difficile à tester et à réutiliser, car ceci implique de considérer tous les autres modules qui sont appelés;

- La métrique d'instabilité d'un module est représentée par le ratio entre le couplage C_e et la somme de toutes les dépendances du module. Elle est exprimée par la formule suivante : $I = C_e / C_e + C_a$ où le paramètre I dénote la valeur d'instabilité du module respectif; si la valeur I approche de 0, ceci indique que le module réalisé est plus stable. En conséquence, il sera plus difficile d'adopter les changements;

- La métrique de couverture des tests de système est calculée par le pourcentage des méthodes ou modules sous l'activité de test. Cette mesure est employée pour la performance du test logiciel dans chaque itération respective; elle nous assure de

produire une suite de jeux de test plus fiable et plus complète afin de valider l'artefact construit de manière fiable et de détecter au bon moment les problèmes apportés par les modifications. En pratique, la décroissance continuelle des valeurs de mesure donne l'alerte que moins d'effort est mis dans le test du logiciel de l'itération; le projet risque d'avoir les défauts non détectés dans le code, des actions correctives doivent être mises en place.

3.4.5 Conclusion

À partir des revues [67; 72; 79], nous apportons les points de vues constatés suivants.

- La difficulté majeure de la gestion de projet réside dans le contrôle des changements intervenus au cours du projet. Ceci exige la souplesse de la maîtrise de projet pour être adaptatif dans un environnement de développement instable; ce dernier est influencé par des facteurs variés incertains tels que les besoins des clients sont modifiées fréquemment, les erreurs ou les enjeux sont identifiés et débogués tardivement et des membres de l'équipe de développement ont mal collaboré, etc. Ces facteurs provoqueront des problèmes comme des efforts excessifs pour remanier, la baisse de productivité de l'équipe, le retard de livraison du produit et la détérioration de la qualité du système non conforme aux exigences des clients. L'évolution des modèles de développement montre une tendance telle que le développement progresse de manière incrémentielle et itérative, dans laquelle l'accent est mis sur l'interaction avec le client, l'évaluation du risque et l'intégration continue, la minimisation du cycle de développement. Le projet est piloté par le test des fonctions des modules réalisés; tout ça rend le projet plus contrôlable en ayant plus d'agilité pour s'adapter aux changements en cours de projet;

- De multiples métriques sont adoptées dans la gestion de projet Agile de manière pertinente. Elles sont utilisées conjointement afin de fournir une meilleure visibilité du processus tout en mesurant le statut réel de la progression du projet, évaluant la qualité d'artefact et l'efficacité de l'équipe de développement dans chaque itération. Cependant, le profit commercial est considéré comme la mesure primaire et clef afin de planifier le projet et de mesurer la progression. Il affectera toutes les dispositions décisionnelles au cours du projet. Les métriques auxiliaires, en mesurant la qualité du code ou la performance de l'équipe, peuvent aider à l'identification des facteurs qui

limitent et baissent l'efficacité du développement à chaque itération; ces métriques ont pour rôle de faciliter itérativement l'amélioration du processus afin d'atteindre l'objectif commercial de l'organisation.

CONCLUSION

Dans ce mémoire, nous avons présenté et analysé un ensemble de métriques et de méthodes appliquées durant la construction du logiciel. Concrètement, elles se divisent en trois approches principales :

- Les métriques et les méthodologies de l'estimation du logiciel dans les phases préliminaires du développement du logiciel. Diverses mesures ont été proposées pour estimer le projet en termes de son envergure et de l'effort de son développement; elles visent à réduire les imprécisions d'estimation en adoptant des méthodes comme le jugement d'experts, les modèles de régression linéaire, le modèle COCOMO, la méthode des points de fonction et la matrice de jugement, etc. Bien que ces modèles d'estimation offrent une prédiction du projet plus précise et plus fiable, le processus de construction est influencé par des facteurs imprévisibles tels que le choix du langage de développement, la communication au sein de l'équipe, le type et la complexité du projet, etc. Également, les écarts entre les affirmations subjectives sont inévitables pendant l'estimation du logiciel pour déterminer les paramètres d'entrée des modèles d'estimation de l'effort, la similitude du projet réel et des projets historiques, etc. On constate que l'activité d'estimation du logiciel est nécessaire et importante durant le développement, car elle fournit des lignes de référence comme critères de comparaison tout en reflétant la progression de la construction. Elle devra être calibrée au fur et à mesure de la construction en éliminant graduellement les incertitudes et les imprécisions.

- Les métriques appliquées au perfectionnement du processus de développement. Ces métriques ont été inventées surtout pour piloter le contrôle de la construction et assurer la qualité du produit final. Elles sont groupées en deux

catégories principales : *les métriques du produit logiciel* qui capturent et quantifient les caractéristiques de l'objet logiciel en reflétant la taille du logiciel, la complexité de l'architecture du système et la structure du flux de données du système. Ces mesures aident à mieux comprendre la conception du code développé, à évaluer efficacement la qualité de l'artefact, à faciliter une prise de décision convenable. *Les métriques du processus logiciel* jouent des rôles d'indicateurs significatifs pour superviser la progression de la construction en termes de temps prévu et d'effort consacré, et analyser la qualité et l'efficacité du processus et la performance de l'équipe de développement. Ce genre de mesure comprend les mesures comme la densité de défauts du code, le taux de correction des défauts et la productivité de l'équipe, etc. Basé sur les études vues dans le deuxième chapitre, on affirme que le volume logiciel et la complexité de la structure logicielle sont fortement corrélés avec l'effort de développement, les attributs qualitatifs du produit final comme les fonctions, l'extensibilité et la facilité de maintenance;

- Les mesures et les mécanismes qui permettent de suivre la progression de la construction en termes de temps prévu, d'effort de construction et de qualité du produit logiciel. La progression est reflétée par le pilotage du projet basé sur des jalons prédéfinis, les courbes référentielles des projets similaires et achevés ou les objectifs qualitatifs préétablis. Ces mesures aident la gestion du projet à prendre les décisions de manière pertinente et adéquate pour identifier et corriger les erreurs ou les enjeux le plus tôt possible, prévenir les risques potentiels, calibrer la planification du projet en réassignant les affectations des ressources selon la complexité des tâches à achever, etc. Les différents modèles de développement ont un impact effectif sur la qualité du processus de développement et l'efficacité de l'équipe; il existe des métriques spécifiques et pertinentes pour chaque modèle de développement ayant leurs propres caractéristiques. Aujourd'hui, le développement d'un projet informatique tend à être plus grand et plus complexe et on rencontre fréquemment au cours du développement des problèmes comme les changements des exigences des

clients, l'identification tardive des erreurs pendant l'intégration des modules et l'effort supplémentaire de remaniement pour corriger le tir. Les modèles Agiles fournissent la souplesse, la rapidité et l'efficacité pour adopter les changements intervenus pendant les itérations du développement; la progression des projets Agiles est présentée par le profit commercial apporté par chaque fonction réalisée du système. De plus, un ensemble de métriques auxiliaires est également utilisé durant chaque itération afin d'identifier les risques potentiels ou les erreurs survenues; elles facilitent le perfectionnement du processus itération par itération.

Malgré le grand nombre de métriques diverses ou de mesures théoriques préconisées pour le développement, le domaine de la mesure du logiciel n'est pas encore mûr. Tout d'abord certaines métriques ne sont pas définies et utilisées de façon consistante par les différentes organisations industrielles ou les chercheurs académiques. Ces derniers utilisent différentes définitions ou règles de comptage incohérentes avec leurs propres objectifs d'utilisation de la mesure; le nombre de lignes de code en est un exemple vu à la section 2.2.2. Ensuite, la sélection et l'adaptation des métriques appropriées parmi celles qui existent se fait en fonction des différents contextes de construction de projet; ces derniers peuvent comprendre de multiples buts de mesure au cours de la construction, la structuration de l'équipe de développement, les différents niveaux d'exigence de qualité du système final, etc. Ceci est dû au fait que chaque contexte de développement correspond à certaines métriques spécifiques qui seront plus adaptées et plus indicatives, il sera donc très difficile d'établir un ensemble des métriques standard pour répondre aux multiples objectifs de mesure. Afin de procurer un choix des métriques plus raisonnable et plus complet, nous faisons les constatations suivantes selon les différents contextes de construction.

- **Appliquer les métriques en fonction des différents buts de mesure**. Certaines métriques et méthodes de mesure sont plus utiles à une phase spécifique de la construction : dans les phases préliminaires comme l'analyse des exigences et la conception du système, la prédiction du projet à réaliser est l'activité centrale de l'équipe de développement et les entités mesurables sont les documents de spécification du projet et de conception du système. Le nombre de lignes de code et

de points de fonction sont des mesures applicables dans ces phases pour estimer le volume du projet; on doit mentionner ici que les points de fonction sont plus adaptés et plus robustes que les lignes de code, parce que la règle de comptage est indépendante des langages de programmation employés, mais la méthode demande des estimateurs avec expérience. Plusieurs techniques sont disponibles pour estimer l'effort et le coût du projet incluant les modèles de régression linéaire, le modèle COCOMO et le jugement d'experts qui sont basés sur des projets achevés et similaires; la complexité de McCabe est une autre métrique performante afin de prédire la complexité à partir de la conception du système en identifiant les modules qui auront le plus d'erreurs. Dans la phase d'implémentation, le contrôle de la construction est l'activité principale; les métriques appliquées permettent de capturer le statut réel du projet de construction afin d'en mesurer la progression. Les métriques comme la date, l'effort appliqué et le coût sont souvent comparées avec les valeurs planifiées; les méthodes incluant les jalons et la technologie EVM permettent de poursuivre l'avancement de la construction. Dans la phase des tests d'artefacts implémentés, des métriques touchant le nombre de défauts et la densité de défauts sont adoptées pour évaluer la fonction de l'artefact réalisé.

- **Appliquer les métriques pertinentes selon les différents exigences des objectifs de qualité du produit final**. Des métriques différentes sont disponibles pour juger chaque aspect qualitatif d'un artefact logiciel. Si on évalue la convivialité du logiciel, par exemple, le degré de satisfaction des clients en termes de facilité d'utilisation est la mesure la plus parlante. Autrement, la densité des commentaires et les métriques de complexité de la conception produiront les données les plus indicatives pour mesurer la facilité de maintenance. Si l'accent est mis sur la fiabilité du système, des métriques telles que la densité de défauts, le nombre ou le pourcentage de jeux de test réussis ou échoués sont des choix appropriés.

- **Appliquer les métriques en vue des différents rôles des membres au sein de l'équipe de développement**. Les individus jouent des rôles différents dans l'équipe et ont des intérêts différents dans la collection des données : les développeurs individuels choisissent des métriques comme l'effort contribué au projet, la durée

estimée pour leur tâche, la complexité du module sur lequel ils travaillent, le nombre de lignes de code produit par eux-mêmes et la densité de défauts. Le chef de projet mettra l'emphase sur la complexité de McCabe pour les modules de la conception du système et le volume estimé du projet afin de planifier les tâches des individus, la date et l'effort prévus, ainsi que les valeurs réelles qui sont obtenues par le suivi pendant la construction. Également, le nombre de défauts trouvés lors de l'intégration des modules, la densité des défauts et le pourcentage de jeux de test réussis permettent de connaître la fonction et la fiabilité du système réalisé. Finalement, le département de marketing s'intéresse plutôt aux métriques incluant la valeur commerciale du produit final en fonction de l'investissement avant le projet, le degré de satisfaction des clients, la comparaison entre le temps prévu et le budget disponible et les valeurs obtenues au cours de la construction.

Dans la perspective du futur, l'utilité et l'intégrité des mesures proposées dans ce mémoire doivent être toujours considérées et validées selon les types de projet et les divers environnements de développement de la pratique industrielle.

BIBLIOGRAPHIE

[1] Colin Kirsopp. 2001. Measurement and the software development process. London, UK, the proceedings of the 12th ESCOM (European Software Control and Metrics Conference). Shaker Publishing

[2] Software Productivity Center. 2002. Eight Step Metrics Program. www.spc.ca/resources/metrics/index.htm

[3] Pierre Bourque, Robert Dupuis. 2004. Software engineering body of knowledge. IEEE Computer Society

[4] Jaak Jurison. 1999. Software project management: The manager's view. Communications of the Association for Information Systems, Vol. 2, No. 17

[5] Kjetil Molokken, Anette C. Lien, Magne Jorgensen, Sinan S. Tanilkan, Hans Gallis, Siw E. Hove, Frank Bomarius, Hajimu Lida. 2003. Does use of development model affect estimation accuracy and bias? Congress Product Focused Software Process Improvement (PROFES) 2004, Vol. 3099, pp. 17-29. Springer Publishing

[6] Kjetil Molokken, Magne Jorgensen. 2005. A comparison of software project overruns --- flexible versus sequential development models. IEEE Transactions on Software Engineering, Vol. 31, No. 9, pp. 754-766

[7] Patricia J. Guinan, Samer Faraj. 1998. Reducing work related uncertainty: the role of communication and control in software development. Proceedings of the Thirty-First Hawaii International Conference, Vol. 6, pp. 73-82

[8] Greg L. Stewart, Murray R. Barrick. 2000. Team structure and performance: assessing the mediating role of intra team process and the moderating role of task type. Academy of Management Journal, Vol. 43, part 2, pp. 135-148

[9] O. Mizuno, T. Kikuno, K. Inagaki, Y. Takagi, K. Sakamoto. 2000. Statistical analysis of deviation of actual cost from estimated cost using actual project data. Information and Software Technology, Vol. 42, No. 7, pp. 465-473. Elsevier Science

[10] Katrina D. Maxwell, Luk Van Wassenhove, Soumitra Dutta. 1996. Software development productivity of European space, military, and industrial applications. IEEE Transaction On Software Engineering, Vol. 22, No. 10, pp. 706-718

[11] Lutz Prechelt. 2000. An empirical comparisons of seven programming languages. IEEE Computer Society, Vol. 33, No. 10, pp. 23-29

[12] Wayne C. Lim. 1994. Effects of reuse on quality, productivity, and economics. IEEE Software, Vol. 11, Issue 5, pp. 23-30, IEEE Computer Society Press

[13] Frank Padberg. 2001. Tracking the impact of design changes during software development. ICSE-Workshop on Economics-Driven Software Engineering Research 3, pp. 50-55

[14] K Kavoussanakis, Terry Sloan. 2001. UKHEC Report on Software Estimation. Technical Report, UKHEC

[15] Kathleen Peters. 1999. Software project estimation. Software Productivity Centre inc. Vancouver, BC. www.spc.ca

[16] R. Agarwal, Manish Kumar, Yogesh, S. Mallick, R.M. Bharadwaj, D. Anantwar. 2001 Estimating software projects. ACM SigSoft Software Engineering Notes. Vol. 26, issue 4, pp. 60-67, ACM Press

[17] Alain Abran, Pierre N. Robillard. 1996. Function point analysis: an empirical study of its measurement processes. IEEE Transactions on Software Engineering, Vol. 22, No. 12, pp. 895-910, Institute of Electrical and Electronics Engineers, NY, USA

[18] Hareton Leung, Zhang Fan. 2002. Software cost estimation. The Hong Kong Polytechnic University

[19] Serge Oligny, Pierre Bourque, Alain Abran, Bertrand Fournier. 2000. Exploring the relation between effort and duration in Software engineering. World Computer Congress, pp. 1, Beijing China

[20] Daniel V. Ferens.1988. Software size estimation techniques. Aerospace and Electronics Conference, Vol. 2, pp. 701-705

[21] Luiz A. Laranjeira. 1990. Software size estimation of object-oriented systems. IEEE Transactions on Software Engineering, Vol. 16, No. 5, pp. 510-522

[22] Christopher J. Lokan. 1996. Early size prediction for C and Pascal programs. Journal of Systems and Software, Vol. 32, Issue 1, pp. 65-72

[23] Graham C. Low, D. Ross Jeffery. 1990. Function points in the estimation and evaluation of the software process. IEEE Transactions on Software Engineering, Vol. 16, No. 1, pp. 64-71

[24] J.J. Dolado. 1997. A study of the relationship among Albrecht and MarkII Function Points, lines of code 4GL and effort. Journal of Systems and Software, Vol. 37, Issue 2, pp. 161-173, Elsevier Science

[25] Hee Beng Kuan Tan, Yuan Zhao, Hong Yu Zhang. 2006. Estimating LOC for information systems from their conceptual data model. ICSE (International Conference on Software Engineering) 2006, Shanghai, China, pp. 321-330

[26] Eduardo Miranda. 2000. An evaluation of the paired comparisons method for software sizing. ICSE 2000 (International Conference on Software Engineering), Limerick, Ireland, pp. 597-604

[27] June Verner, Graham Tate. 1992. A software size model. IEEE Transaction On Software Engineering, Vol. 18, Issue 4, pp. 265-278, IEEE Press

[28] Fabrizio Riguzzi. 1996. A survey of software metrics. DEIS Technical Report, LIA series No. 17, Universita degli Studi di Bologna

[29] Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. MacLeod, Michael J. Merritt. 1978. Characteristics of software quality. North Holland

[30] Norman E. Fenton, Shari Lawrence Pfleeger. 1997. Software metrics: a rigorous and practical approach. PWS Publishing Company, Part I, pp. 3-20, Part II, pp. 243-337

[31] S.D. Conte, H.E. Dunsmore, and V.Y. Shen. 1986. Software engineering metrics and models. Menlo Park, Calif.: Benjamin/Cummings, Chapter 4, pp. 183 – 232

[32] I. Sommerville. 2001. Software Engineering, Sixth Edition. Addison-Wesley Publishers Limited.

[33] Steve McConnell. 1996. Rapid development: taming wild software schedules. Microsoft Press.

[34] Ikatura Minoru, Akio Takayanagi. 1982. A model for estimating program size and its evaluation. Proceedings, Sixth International Conference on Software Engineering, Long Beach, CA. IEEE Computer Society Press, pp. 104-109

[35] Jose Javier Dolado. 2000. A validation of the component-based method for software size estimation. IEEE Transactions on Software Engineering, Vol. 26, No. 10, pp. 1006-1021

[36] M. Hakuta, F. Tone, M. Ohminami. 1997. A software size estimation model and its evaluation. Systems and software, Vol. 37, pp. 253-263

[37] Jack E. Matson, Bruce E. Barrett, Joseph M. Mellichamp. 1994. Software development cost estimation using function point. IEEE Transactions on Software Engineering, Vol. 20, No. 4, pp. 275-286

[38] Charles R. Symons. 1988. Function point analysis: difficulties and improvements. IEEE Transactions on Software Engineering, Vol. 14, No. 1.

[39] Martin Shepperd, Chris Schofield. 1997. Estimating software project effort using analogies. IEEE Transactions on Software Engineering, Vol. 23, No. 12.

[40] Paul Goodman. 2004. Software metrics: best practices for successful IT management. Rothstein Associates Company, Chapitre 5.3

[41] John Roche, Mike Jackson, Martin Shepperd. 1994. Software measurement methods: an evaluation and perspective. Assessment of Quality Software Development Tools Proceedings, Vol. 3, Issue 7-9, pp. 50-69

[42] Steve McConnell. 2003. Code complete: A practical handbook of software construction. Microsoft Press, Second edition.

[43] Norman Fenton. 1999. Software metrics: successes, failures and new directions. The Journal of Systems and Software, Vol. 47, pp. 149-157

[44] Bradford L. Goldense. 1997. Motivator & metrics for product development. Electronics Industries Forum of New England, Professional Program Proceedings, pp. 67-83

[45] Michael K. Daskalantonakis. 1992. A pratical view of software measurement and implementation experiences within Motorola. IEEE Transactions on Software Engineering, Vol. 18, No. 11.

[46] Eight-Step Metrics Program: Step 3 --- Define metrics required to reach goals. Software Productivity Centre Inc. Vancouver, BC. www.spc.ca

[47] William R. Duncan. 2000. A guide to the project management body of knowledge. Third Edition, Project Management Institute.

[48] Christof Ebert. 1999. Technical controlling in software development. International Journal of Project Management, Vol. 17, No. 1, pp. 17-28

[49] Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering, Second edition, Addison Wesley Publishing. Chapters 9, 10, 11.

[50] Laurie Honour Werth. 1993. Introduction to Software Process Improvement. Lecture Notes on Software Process Improvement, CMU/SEI-93-EM-8, Software Engineering Institute, Carnegie Mellon University

[51] Rosenberg, Jarrett. 1997. Some misconception about lines of code. 4th International Software Metrics Symposium Proceedings; pp.137-142

[52] Christopher Lokan, Alain Abran. 1999. Multiple viewpoints in functional size measurement. International Workshop on Software Measurement; pp. 121-132

[53] Linda M. Laird, M. Carol Brennan. 2006. Software measurement and estimation: a pratical approach. John Wiley & Sons,.; Chapter 5, pp. 55-78; Chapter 10, pp. 181-195

[54] Ryouei Takahashi. 1997. Software quality classification model based on McCabe's complexity measure. Journal of System Software; Vol. 38, pp. 61-69

[55] Linda H. Rosenberg. 1998. Applying and interpreting object oriented metrics. Software Technology Conference.

[56] Taghi M. Khoshgoftaar, John C. Munson. 1990. Predicting software development errors using software complexity metrics. IEEE Journal on Selected Areas in Communications. Vol. 8, No. 2.

[57] Joel Troster. 1992. Assessing design-quality metrics in legacy software. Conference of the Centre for Advanced Studies on Collaborative Research. Vol. 2, pp. 113-131.

[58] Nerville I. Churcher, Martin Shepperd. 1995. Towards a conceptual framework for object oriented software metrics. ACM SIGSOFT, Software Engineering Notes, Vol. 20, No. 2.

[59] Shyam R. Chidamber, Chris F. Kemerer. 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, Vol. 20; No. 6.

[60] Ho Leung Tsoi. 1999. A framework for management software project development. The Proceedings of the 1999 ACM symposium onApplied computing, pp. 593-597.

[61] Ken Shumate, Terry Snyder. 1994. Software project reporting: management, measurement, and process improvement. Proceeding of the Conference TRI-Ada'94, pp. 41-45. ACM Press.

[62] George Stark, Robert C. Durst, C.W. Vowell. 1994. Using metrics in management decision making. IEEE Computer Publication, Vol. 27, Issue 9: pp. 42-48

[63] Bob Hunt. 2007. Parametric project monitoring and control: performance-based progress assessment and prediction. Aerospace Conference 2007 IEEE, pp. 10-21

[64] Walt Lipke. 2006. Schedule is different. http://www.earnedschedule.com

[65] Craig Larman, Victor R. Basili. 2003. Iterative and incremental development: a brief history. IEEE Computer Society Press, Volume 36, Issue 6, pp. 47-56

[66] Elaine L. May, Barbara A. Zimmer. 1996. The evolutionary development model for software. Article No.4, HP Journal

[67] Michael Coram, Shawn Bohner. 2005. The impact of agile methods on software project management. Engineering of Computer-Based Systems, Volume 68, Issue 4-7, pp. 363-370

[68] Boehm, B.W. 1988. A spiral model of software development and enhancement. IEEE Computer, Volume 21, Issue 5, pp. 61-72

[69] Robert C. Martin. 1999. Iterative and incremental development. Engineering Notebook Column, C++ Report.

[70] Trond Johansen, Tom Gilb. 2005. From Waterfall to evolutionary development: how we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market. Published by INCOSE with permission of Tom Gilb and FIRM A/S.

[71] Lisa Brownsword, Jim Smith. 2005. Using earned value management in spiral development. Technical Note, CMU/SEI-2005-TN-016, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA.

[72] Deborah Hartmann, Robin Dymond. 2006. Appropriate Agile measurement: using metrics and diagnostics to deliver business value. Proceedings of the conference on AGILE 2006, pp. 126-134.

[73] Tamara Sulaiman, Brent Barton, Thomas Blackburn. 2006. Agile EVM – earned value management in scrum projects. AGILE 2006, pp. 7-16.

[74] Ron Jeffries. 2004. A metric leading to agility. http://www.xprogramming.com/xpmag/jatRtsMetric.htm

[75] Mishikin Berteig. 2005. A metric leading to success. http://www.agileadvice.com/archives/2005/11/a_metric_leadin.html

[76] Pekka Abrahamsson, Jussi Rokainen, Juhani Warsta. 2002. Agile software development methods: Review and analysis. VTT Publications.

[77] Kirk Knoernschild. 2006. Using metrics to help drive agile software. http://www.agilejournal.com/articles/

[78] Dan Rawsthorne. 2006. Calculating earned business value for an agile project. http://www.agilejournal.com/articles/

[79] Liz Barnett. 2007. Being Agile in 2007. http://www.agilejournal.com/articles/