UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MESURE DE LA QUALITÉ DES ESTIMATIONS DU PROGICIEL D'ESTIMATION SLIM

PROPOSITION DE MÉMOIRE

PRÉSENTÉE COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE DE GESTION

PAR

IPHIGÉNIE NDIAYE

JUIN 2000

TABLE DES MATIÈRES

CHAPITRE I

DÉFINITION

1.1 MOTIVATION

La prise de décision n'est pas une tâche facile; c'est plus qu'un défi. Elle est d'une difficulté inhérente qui est aggravée par la complexité et l'allure rapide des changements qui caractérisent le génie logiciel. Des décisions critiques ayant un impact sur le succès du projet ou même sur l'organisation entière doivent être prises rapidement, sur la base d'une information limitée au point d'être insuffisante ou d'une information abondante qui est difficilement maniable (Dion et Abran, 1999).

Les gestionnaires du développement de logiciel doivent néanmoins prendre des décisions importantes, dans un contexte où l'inconnu existe, tel que l'estimation a priori. En effet, l'estimation de projet au début du cycle de développement est un élément déterminant du processus décisionnel des investissements en logiciel. Cependant, à mesure que progressent les projets, les difficultés spécifiques aux modèles d'estimation sont progressivement éliminées de sorte qu'à la fin des projets, il n'existe plus d'inconnues, d'incertitudes, ni de risques (Abran et Robillard, 1993). Ce qui rend l'estimation plus facile, mais qu'en est-il du processus décisionnel? Ainsi, la motivation principale dans ce travail est d'améliorer la qualité de la prise de décision d'estimation des gestionnaires de projets informatiques.

1.2 OBJET

Plusieurs modèles pour estimer l'effort requis pour développer un système informatique, à partir des paramètres comme la taille estimée des différents types de projets, la composition de l'équipe de développement, le type de langage utilisé et d'autres variables, sont décrits dans la littérature (Bourque, 1988). Cependant, cette recherche ne portera que sur un seul type de modèle de productivité. Il s'agit du progiciel SLIM (*Software Life-Cycle Model*) qui

utilise les lignes de code comme unité de mesure. Et comme certaines organisations utilisent ce progiciel, il est donc important pour ces dernières de connaître son degré de fiabilité.

1.3 OBJECTIF

L'objectif principal de cette recherche est d'étudier la fiabilité du modèle de productivité SLIM, c'est-à-dire évaluer la qualité des estimations faites avec ce progiciel.

1.4 UTILISATEURS

Plusieurs études montrent que plus de deux tiers des projets de développement informatiques dépassent très largement leurs délais, ce qui engendrent des pertes considérables pour les entreprises. La décision de commencer ou de continuer un projet est souvent influencée par les estimations d'effort de développement (Stroian, 1999). Ainsi, les principaux utilisateurs de cette recherche sont les gestionnaires en informatique plus particulièrement les chefs de projets et les décideurs.

Ce travail est également destiné aux praticiens qui manipulent les progiciels pour estimer les coûts en développement de logiciel.

1.5 DOMAINE

Cette recherche réside dans le domaine du MIS (*Management Information Systems*) plus particulièrement tout ce qui touche le développement du logiciel et les modèles de productivité a posteriori plutôt connus sous le nom de modèles d'estimation des coûts logiciels.

1.6 ÉTENDUE

L’échantillon utilisé pour effectuer la recherche est la sixième version de la base de données ISBSG (*International Software Benchmarking Standard Group;* release 6). Elle est la première grande base de données des projets informatiques disponibles pour les praticiens et les chercheurs.

CHAPITRE II

PROBLÉMATIQUE

D'après les rapports de praticiens et les résultats de plusieurs recherches (Heemstra, 1992), les projets logiciel ne sont pas sous contrôle de façon régulière et invariablement, l'effort fourni lors du développement dépasse l'effort estimé, résultant en une livraison tardive du logiciel. Il n'y a aucun doute que l'estimation du logiciel est un problème pour la gestion d'un projet logiciel. Au premier coup d'œil, les questions à répondre à ce sujet sont simples : combien de temps et d'effort cela coûtera-t-il pour développer un logiciel? Quels sont les facteurs de coûts dominants? Quels sont les facteurs importants de risques? Cependant, les réponses à ces questions ne sont ni simples, ni faciles.

Pour cela, plusieurs progiciels d'estimation de coûts logiciel sont disponibles et sont équivalents aux modèles «*Software Cost Estimation* » (SCE). Ces progiciels sont une des techniques que la gestion de projet peut utiliser pour estimer l'effort et la durée du développement du logiciel.

2.1 LES PRINCIPES ET LES MODÈLES D'ESTIMATION DES COÛTS LOGICIELS (Heemstra, 1992)

La plupart des modèles actuels sont des modèles à deux étapes comme le décrit la figure 1. La première est la mesure de la taille et la deuxième fournit un facteur d'ajustement à l'estimation nominale de l'effort.

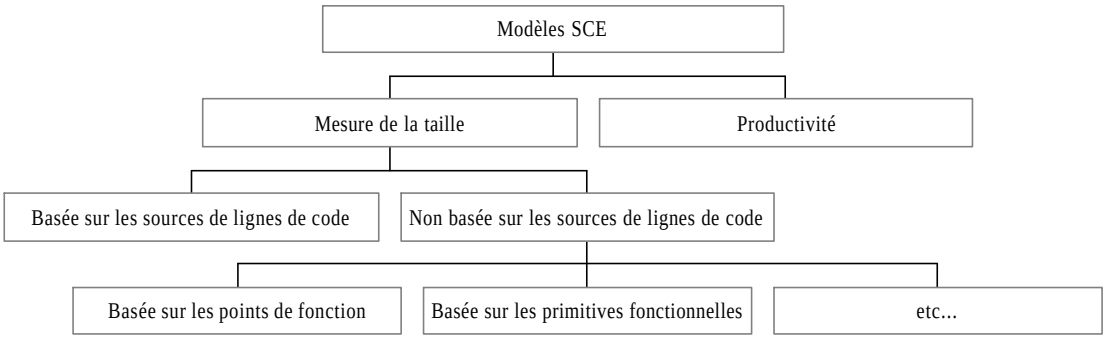

Figure 2.1: Structure des modèles d'estimation des coûts logiciels (SCE)[1]

Dans la première phase, une estimation concernant la taille du produit à développer est obtenue. En pratique, plusieurs techniques de mesure de la taille sont utilisées telles que les points de fonction et les lignes de code (LOC). Mais, d'autres techniques comme la « science logique » (Boehm, 1981; Van Genuchten, 1991) et la méthode de « Bang de DeMarco » (DeMarco, 1982, 1984; Stutzke, 1997) ont été définies . Le résultat de ces méthodes est la taille/le volume du logiciel à développer, exprimée en de lignes de code, le nombre d'états ou de rapports, ou le nombre de points de fonction.

Dans la deuxième phase, le temps et l'effort nécessaire au développement du logiciel sont estimés. Premièrement, l'estimation de la taille est convertie en une estimation de l'effort nominal en mois/personne.

Vu que cet effort nominal ne tient pas compte des caractéristiques spécifiques connues du produit logiciel, de la façon dont ce dernier sera développé, des moyens de productions, un nombre de facteurs influençant les coûts (*cost-drivers*) sont ajoutés au modèle. L'effet de ces *cost-drivers* peut être estimé; et cet effet est appelé le facteur d'ajustement à l'estimation nominale de l'effort. L'application de ce facteur d'ajustement fournit une estimation plus réaliste.

---

[1] Source : Heemstra, F. J., 1992, « Software Cost Estimation », *Information and Software Technology*, Guildfor, Octobre, p. 627-639.

L'effort et la durée d'un projet sont estimés pour permettre aux gestionnaires de déterminer les mesures importantes d'affaires telles que le coût du produit, le retour sur investissement (ROI) et le temps de vente. Cependant le processus d'estimation ne s'avère pas évident pour plusieurs raisons (Stutzke, 1997).

La première est que les projets doivent souvent satisfaire des buts incompatibles. Les logiciels à développer (ou à maintenir) doivent fournir une fonctionnalité spécifique dans un délai, un coût et des critères de performances spécifiques, et avec un certain niveau de qualité désiré. Ces multiples contraintes compliquent alors le processus d'estimation (Stutzke, 1997). Ainsi, lors de la planification de développement d'un nouveau produit logiciel, une compatibilité pourrait être déterminée entre le coût, la durée et la taille. Par exemple, un coût moindre nécessiterait une réduction de la taille du produit (Gaffney et Cruishank, 1997).

La deuxième raison est que les estimations sont nécessaires avant que le produit soit bien défini. La fonctionnalité du logiciel est difficile à définir, spécialement dans les premières étapes du projet. En général, la précision des estimations est proportionnelle à l'évolution du projet, c'est à dire qu'elle augmente au fur et à mesure que les informations concernant le produit sont disponibles, comme sa structure, sa taille et la productivité de l'équipe de développement.

Le besoin de modifier le code existant est la troisième raison de la difficulté de l'estimation du coût d'un logiciel. Il n'est pas facile d'identifier et de quantifier les facteurs affectant l'effort nécessaire pour incorporer le code actuel dans un produit. Le code doit être localisé, compris, modifié, intégré et testé. Le coût de ces activités dépend de la façon dont le code est structuré, de la connaissance du programmeur en ce qui concerne ce code, et de bien d'autres choses. La modification du code est un sous-ensemble de la grande activité nommée la réutilisation du logiciel qui fait face à ces mêmes problèmes.

La quatrième raison est le changement de la manière de construire un logiciel. De nouveaux processus de développement ont émergés en 1997 (Stutzke) et nécessitaient de nouvelles

approches d'estimation du coût et de la durée. Ces processus s'efforcent de fournir une meilleure qualité des logiciels, de produire des logiciels plus modulaire et qui l'on peut maintenir, et/ou de livrer le produit plus rapidement à l'utilisateur final. Pour atteindre ces objectifs, les développeurs utilisent des combinaisons de composants de code préétablis et des outils qui allègent le travail.

La figure 2 montre les différentes étapes à suivre dans le contexte de l'estimation du coût général (Heemstra, 1992); Il s'agit tout d'abord de la mesure de la taille du logiciel en déterminant les métriques de la taille de ce logiciel et d'évaluer ensuite les différents facteurs de productivité dans la phase suivante où tous les facteurs influençant le coût sont déterminés. En plus de ces deux composants, une phase de distribution et une autre d'analyse de la sensibilité et du risque sont distinguées.

Dans cette phase de distribution, la composante de l'effort total et de la durée est ventilée à travers les phases et les activités du projet. Cette division devrait être basée sur des données empiriques des projets antérieurs.

La phase de l'analyse de la sensibilité et du risque supporte la gestion de projet – spécialement au début du projet quand l'incertitude est grande – en déterminant les facteurs de risque du projet et la sensibilité des estimations aux ensembles de facteurs influençant les coûts.

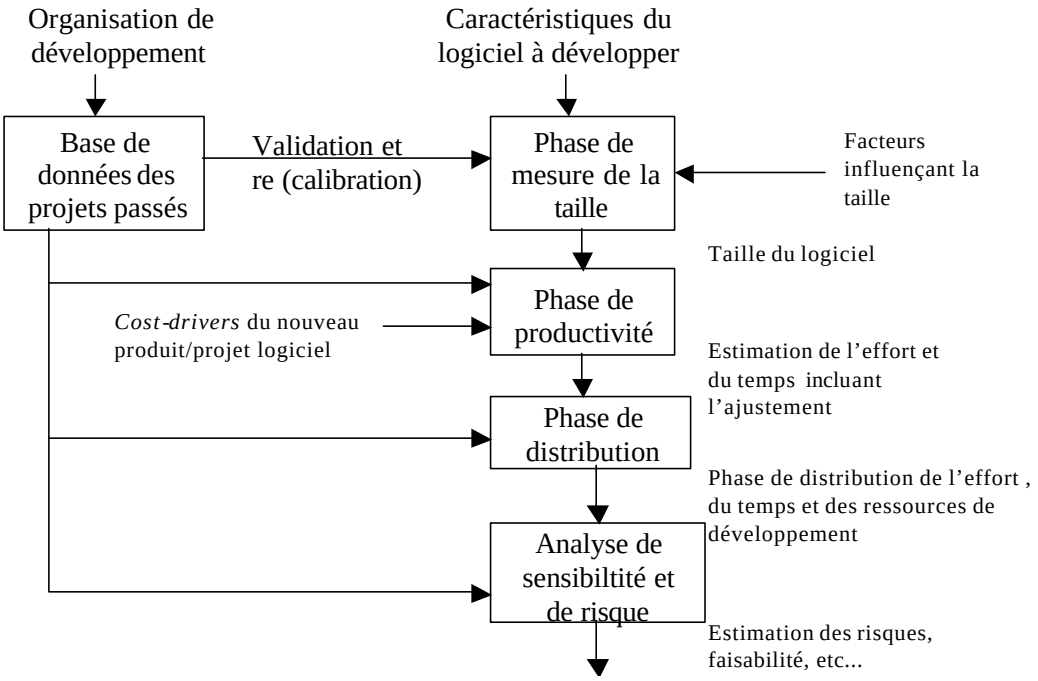

Figure 2.2: Structure générale de l'estimation du coût[2]

Qui plus est, il y a deux classes de base des méthodes d'estimation. Il s'agit tout d'abord de l'estimation basée sur l'expérience qui peut être imparfaite à cause de la désuétude des données historiques utilisées. La deuxième classe est celle des modèles paramétriques qui ont typiquement une perspective particulière. Certains concordent avec le processus de développement standard militaire (tel que défini par le département de la défense américaine). D'autres modèles concordent avec les procédures de développement commerciales. Les estimateurs doivent choisir les modèles qui vont avec l'environnement de leur projet et s'assurer que ces modèles sont correctement calibrés à cet environnement. En dépit de leurs faiblesses intrinsèques, toutes ces deux classes de méthodes ont leur usage (Stutzke, 1997).

[2] Source : Heemstra, F. J. 1992, « Software Cost Estimation », *Information and Software Technology*, Guildfor, Octobre, p. 627-639.

L'estimation de l'effort et de la durée du développement du logiciel est devenue un sujet d'une importance croissante. Il arrive souvent que le logiciel coûte plus que ce qui a été estimé et qu'il soit livré plus tard que prévu (Heemstra, 1992). Ainsi, il est important que le temps nécessaire au développement du produit logiciel soit estimé avec précision et qu'un compromis soit fait entre la durée et l'effort de développement. La durée est alors un paramètre à considérer lors de la planification d'un projet de développement de logiciel. Sa compression a des effets sur l'effort de développement. De ce fait une très grande quantité d'effort appliquée au développement du logiciel dans une court intervalle de temps n'est pas faisable (Gaffney, 1997).

De plus, il s'avère que la plupart des logiciels ne répondent pas à la demande (Heemstra, 1992). La variation des exigences de qualité du produit a un impact sur la durée et/ou sur le coût. La plupart du temps, les estimateurs peuvent s'assurer de la plus grande qualité du produit par plusieurs tests. Conséquemment, cela peut accroître l'effort et le temps de développement au-delà de la durée nécessaire pour un produit logiciel qui n'exige pas ce niveau de qualité (Gaffney, 1997).

Des études montrent que près de deux tiers des projets dépassent très largement leur délais (Garmus et Herron, 1996; Ingram, 1994, Jones, 1997; Lederer et Prasad, 1993). Les projets sont généralement en retard sur la date de livraison de 25 à 50%, et l'importance de ce retard est proportionnelle à la taille du projet (Jones, 1994)

En ce qui concerne l'effort de développement, il appert que 63% des activités auraient besoin de moins d'effort que ce qui est estimé et que 14% auraient besoin de plus d'effort qu'estimé (Walkerken et Jeffery, 1996). Les raisons des écarts entre les estimations et la réalité se révèlent spécifiques pour le développement. Dans les organisations où les mesures sont prises, les raisons sont principalement reliées à la sous-estimation de la quantité de travail, de la complexité de l'application et aux spécifications qui se révèlent irréalistes du point de vue technique (Heemstra, 1992).

Les coûts sont sous-estimés, lorsque certains projets sont entrepris avec une impression exagérée de leur valeur pour la firme étant donnée les coûts de développement (Vicinanza et al., 1990). Subséquemment, les projets jugés de valeur à l'origine pourraient se révéler imprévisibles. Une étude effectuée par Standish Group en 1995 (The Standish Group, 1998) aux États-Unis, montre que 31% des projets ont été arrêtés et 53% ont un dépassement du budget ou sont en retard ou ont livré moins de fonctionnalités que prévues initialement. Seulement 16% finissent leur projet à temps et à l'intérieur du budget; ce qui représente moins de un projet sur six (Stroian, 1999).

Les projets sous-estimés achevés sont généralement livrés prématurément pour respecter le budget, omettant ainsi de tester les caractéristiques importantes et le système lui-même, et résultant en des systèmes incomplets et non fiables (Kemerer, 1989; Kitchenham, 1998).

D'un autre côté, la surestimation d'un projet crée également bien des problèmes. Les surestimations du projet peuvent actuellement hausser le coût de ce dernier en mettant moins de pression sur les programmeurs à être productifs (Abdel-Hamid et Madnick, 1991). De plus, les projets possédant un réel potentiel de profit peuvent être faussement rejetés, résultant en un coût d'opportunité manqué à créer de la valeur au niveau de la firme.

Par conséquent la surestimation et la sous-estimation peuvent engendrer des erreurs coûteuses (Heemstra, 1992), en entraînant une réduction de la productivité moyenne et en augmentant l'effort total (Walkerken et Jeffery, 1996). L'estimation exacte du projet peut réduire ces coûts inutiles et ainsi accroître l'efficience de la firme de même que son efficacité.

C'est alarmant, inquiétant qu'il soit si difficile aux organisations de contrôler le développement du logiciel. Cela est une raison suffisante pour insister sur le fait que l'estimation du coût de développement du logiciel et son contrôle devrait prendre leur place en tant qu'une branche pleinement naissante à l'intérieur d'une discipline qu'est le développement du logiciel.

2.2 QU'EST CE QUI REND L'ESTIMATION DU COÛT DU LOGICIEL DIFFICILE? (Heemstra, 1992)

C'est la question principale posée lorsque les problèmes ci-dessus sont mentionnés; il y a plusieurs raisons, et sans entrer dans les détails, certains sont listés subséquemment :

- Il y a un manque de données des projets logiciels complétés. Ce genre de données pourrait supporter la gestion des projets dans leur phase d'estimation.

- Les estimations sont souvent faites précipitamment sans une appréciation de l'effort nécessaire pour un résultat crédible. De plus, c'est souvent le cas lorsque l'estimation est nécessaire avant que des spécifications claires des exigences du système soient définies. Donc, une pression est faite sur les estimateurs pour qu'ils fassent rapidement une estimation d'un système qu'ils ne comprennent pas vraiment.

- Des spécifications claires, complètes et fiables sont difficiles à formuler, spécialement au début du projet. Des changements, adaptations et ajouts sont plus une règle qu'une exception et conséquemment, les planifications et les budgets doivent être adoptés également.

- Les caractéristiques et le développement du logiciel rendent l'estimation difficile; par exemple, le niveau d'abstraction, de complexité, de mesure du produit et du processus, les aspects innovateurs, etc.

- Un bon nombre de facteurs ont une influence sur l'effort et le temps de développement du logiciel. Ces facteurs sont appelés *cost-drivers*. Des exemples sont la taille et la complexité du logiciel, l'engagement et la participation des utilisateurs dans l'organisation, l'expérience de l'équipe de développement. En général, ces *cost-drivers* sont difficiles à déterminer pendant l'opération.

- Les changements rapides dans la technologie de l'information et la méthodologie de développement du logiciel sont un problème pour stabiliser un processus d'estimation. Par exemple, il est difficile de prévoir l'influence de nouveaux systèmes établis, de quatrième et cinquième langage de génération, des stratégies de prototypages et de bien d'autres.

- Un estimateur (souvent le gestionnaire de projet) peut ne pas avoir beaucoup d'expérience dans le développement des estimations, spécialement pour ce qui est des gros projets.

- Un biais apparent des développeurs de logiciel contribue à une sous-estimation. Il est probable qu'un estimateur ne considère que le temps que devrait prendre une certaine portion du logiciel, et alors qu'il extrapole cette estimation au reste du système,

ignorant ainsi les aspects non-linéaires du développement du logiciel, par exemple la coordination et la gestion.

- L'estimation estime le temps que cela prendrait pour accomplir personnellement une tâche, ignorant le fait que beaucoup de travaux seront faits par des gens moins expérimentés et un personnel junior avec un taux de productivité faible.

- Il existe un sérieux manque d'hypothèses pour ce qui est d'une relation linéaire entre la capacité exigée par unité de temps, et le temps disponible. Cela voudrait dire qu'un logiciel développé par 25 personnes en deux ans pourrait être accompli par 50 personnes en un an. L'hypothèse n'est pas démontrée. Brooks (1975) affirme même le contraire : « ajouter des personnes dans l'exécution des projets en retard ne le rend que plus tardif » (Kitchenham, 1998).

- L'estimateur a tendance à réduire les estimations à un certain degré dans le but de rendre l'offre plus acceptable.

Tous ces problèmes d'estimation du coût logiciel peuvent être adaptés à plusieurs situations. D'une perspective organisationnelle, il existe de nombreuses façons d'améliorer la gestion de projet logiciel telles que l'allocation des responsabilités, la prise de décision, l'organisation du travail du projet, le suivi et la vérification des tâches de développement.

L'estimation du coût logiciel peut également être vue d'un point de vue sociologique et psychologique. Cela réfère à l'engagement des participants, l'organisation d'une cohésion de groupe, le style de leadership.

Le côté technique du travail est également une issue importante à prendre en considération.

Il existe plusieurs facteurs qui ont de l'influence sur l'effort et la durée de développement du logiciel. Plusieurs conditions doivent être remplies pour réduire les problèmes d'estimation du coût logiciel exposés ci-dessus, et pour garantir une base solide pour la prédiction de l'effort, de la durée et la capacité de développer le logiciel (Heemstra, 1992).

2.3 LES PRÉALABLES DE L'ESTIMATION DU COÛT LOGICIEL

Il s'agit :

- d'avoir un aperçu :
    - du produit (logiciel) à développer (quoi);
    - des moyens de production (avec quoi);
    - du personnel de production (qui);
    - de l'organisation de la production (comment);
    - des utilisateurs (pour qui);
- d'avoir la disponibilité
    - de certaines règles générales sur l'estimation du coût logiciel;
    - d'un moyen de structurer les *cost-drivers*, c'est à dire lister les *cost-drivers* les plus importants de chacune de ces différentes catégories : qui, quoi, comment, avec quoi et pour qui;
    - d'une liste des facteurs influents qui sont communément considérés comme importants;
    - des techniques et outils pour l'estimation du coût logiciel.

En ce qui concerne ces techniques, un certain nombre est défini dans la littérature, et différentes suggestions ont été faites pour les classifier (Boehm, 1981; Basili, 1980; Van Genuchten et Koolen, 1991). Les principales à considérer sont une combinaison des techniques primaires suivantes (Boehm, 1981) :

- Les modèles de coûts : ils fournissent une ou plusieurs formules qui produisent une estimation du coût logiciel comme une fonction d'une ou de plusieurs variables;

- L'analogie : cette méthode implique le raisonnement par analogie avec un ou plusieurs projets complétés. Les similarités et les différences entre le nouveau projet et celui complété sont utilisées pour estimer le coût du nouveau projet.

- Le jugement des experts : cette méthode quant à elle implique la consultation d'un ou de plusieurs experts. Lorsqu'il s'agit d'un groupe, les techniques telles que la méthode Delphi peuvent être utilisées pour obtenir un consensus global.

- La technique du haut vers le bas (descendante): une estimation totale du coût d'un projet est dérivée des propriétés globales du produit logiciel. Le coût total est ainsi partagé entre les divers composants.

- La technique du bas vers le haut (ascendante) : un projet logiciel est divisé en ses composants individuels. Chaque composant subit séparément une évaluation de coût, et cela par la personne responsable du développement de ce composant. Les estimations individuelles sont alors additionnées pour donner un coût total.

Bien que cela soit une façon de distinguer les diverses méthodes d'estimation, il est important de réaliser que toutes ces méthodes ont certaines similarités. La principale est qu'elles nécessitent toutes des connaissances historiques d'autres projets logiciels. Cela n'est pas seulement vrai dans le cas évident de l'évaluation du coût par analogie, mais aussi même dans le cas du modèle d'estimation de coût le plus théorique qui doit être validé contre la donnée réelle et calibrée pour des environnements particuliers (Boehm, 1981; Walkerken et Jeffery, 1996; Stutzke, 1997).

Aucune technique particulière d'estimation de coût ne peut parer au besoin d'information accessible et systématique en ce qui concerne le coût et la nature des projets logiciels complétés et toutes les techniques seraient substantiellement améliorées par de telles informations. Ainsi, *une exigence fondamentale* pour toutes ces techniques d'estimation est une base de données historiques qui enregistrent l'information sur le coût du logiciel et sur le projet.

Un autre facteur important est que régulièrement, aucune méthode n'est meilleure que les autres sur tous les aspects; un résumé des forces et faiblesses relatives de ces méthodes est présenté dans le tableau 1 (Boehm, 1981, Van Genuchten et Koolen, 1991; El Emam et al., 1998) :

Tableau 2.1 : Forces et faiblesses des méthodes d'estimation des coûts logiciels[3]

Méthodes	Forces	Faiblesses
Modèles de coûts	• Objectif, répétable, efficient • Permet une analyse de sensibilité • Objectivement calibré à l'expérience • Formule analysable	• Entrées souvent subjectives • Évaluation de circonstances exceptionnelles • Calibré au passé et non au futur
Jugement d'expert	• Peut évaluer la représentativité de l'expérience passée • Peut estimer l'effet de nouveaux facteurs environnementaux et des techniques de production • Peut faire face à des circonstances exceptionnelles	• N'est pas meilleur que la qualité des experts • Sujet à biais • Rappel incomplet
Analogie	• Basée sur des expériences représentatives	• L'expérience passée pourrait ne pas être représentative
Descendante	• « *system level focus* »	• Moins détaillée • Moins stable (Wolverton, 1974)
Ascendante	• Base détaillée • Plus stable due à l'annulation de l'effet de collectivité • Stimule l'engagement individuel	• Peut ignorer les coûts du « *system-level* » • Nécessite plus d'effort

Boehm (1981) montre que les forces et faiblesses particulières sont souvent complémentaires, si bien que la meilleure méthode d'estimation est la combinaison de techniques (Van Genuchten et Koolen, 1991; El Emam et al., 1998). Ainsi, une estimation descendante utilisant le jugement de plusieurs experts basé sur l'analogie peut être comparée à la méthode

[3] Sources : Boehm, B.W. 1981, *Software Engineering Economics*, Prenctice-Hall.

Van Genuchten, M. et H. Koolen 1991. « On the Use of Software Cost Models. » *Information & Management [IFM]*, Vol. 21, no 1, p. 37-44.

Briand, L. C., K. El Emam, et al. 1998. *An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques*. Germany, Fraunhofer Institute for Experimental Software Engineering.

ascendante ayant utilisée le modèle de coût dont les entrées sont fournies par les futurs responsables de l'implémentation du système, dans la continuité du processus itératif.

Une méthode alternative d'estimation des coûts est l'utilisation d'un ou de plusieurs modèles algorithmiques (Vicinenza et al., 1990). Côté, Bourque, Oligny et Rivard (1988) ont identifié plus d'une vingtaine de modèles d'évaluation de l'effort du logiciel dans la littérature incluant COCOMO (Boehm, 1984), Doty (Herd et al., 1977), SLIM (Putnam, 1978), PRICE-S (Frieman et Park, 1979), ESTIMACS (Rubin, 1983) et les points de fonctions (Albrecht et Gaffney, 1983).

En général, les modèles d 'effort algorithmiques utilisent une combinaison des métriques de la taille du logiciel et des facteurs de productivité pour estimer l'effort nécessaire pour compléter le projet. La métrique la plus courante utilisée comme *input* à ces modèles est les lignes de code (LOC); cependant, d'autres métriques de tailles sont également utilisées telles que les points de fonction (Vicinenza, 1990) .

Parmi ces différents modèles, certains sont basés sur des théories du fonctionnement de l'esprit humain pendant le processus de programmation et sur des lois mathématiques que le processus de développement du logiciel est supposé suivre. Et l'un de ces modèles est SLIM (Conte et al., 1986; Stutzke, 1997; Gaffney, 1997).

2.4. APERÇU DU MODÈLE D'ESTIMATION SLIM

Ce modèle de Putnam (1978), est l'un des premiers modèles algorithmiques d'estimation de coût. Il est généralement connu comme un «macro-modèle » d'estimation c'est-à-dire qu'il est approprié aux projets de grande envergure. Il est également basé sur les études de Norden concernant les modèles du cycle de vie des projets logiciels. Putnam utilise la courbe de base Rayleigh conjointement avec un nombre d'hypothèses dérivées empiriquement pour obtenir l'équation suivante (Kitchenham et Taylor, 1984; Putnam et Myers, 1997) :

$$K = \left(LOC / \left(C * t^{4/3} \right) \right) * 3$$

K est l'effort total du cycle de vie en années de travail, t est le temps de développement, LOC représente le nombre de lignes de codes du produit final, et C est le « facteur de technologie », combinant l'effet de l'utilisation des outils, des langages, de la méthodologie et l'assurance de la qualité. Ainsi, cette constante C mesure l'état de la technologie utilisée, l'environnement dans lequel le développement est entrepris, l'équipement de développement disponible et le temps nécessaire pour mettre au point et tester le produit.

En somme, le modèle SLIM décrit le cycle de vie d'un projet logiciel comme étant une courbe Rayleigh, composée d'un certain nombre de sous-cycles conformes au design, au code, au test, à la maintenance, aux activités du projet, etc. (Kitchenham et Taylor, 1984; Putnam et Myers, 1997). De plus, il fait l'hypothèse d'une relation entre la taille du produit, le temps de développement et l'effort total pour un projet particulier. Il peut également être utilisé pour montrer les effets et les limites des compromis entre le temps et l'effort de développement qui représente le coût.

Le modèle SLIM comprend trois composantes qui sont l'estimation , les métriques et le contrôle[4]. Ces trois outils ont chacun leur raison d'être, répondant à des objectifs différents.

[4] http://www.qsma.com

Figure 2.3 : Les composants du progiciels SLIM

SLIM-Estimate : c'est la version basée sur Windows de l'outil d'estimation et de planification SLIM utilisé depuis 1978, pour développer le meilleur plan de projet possible, pour un ensemble de circonstances données. Il a différentes caractéristiques telles que décrites ci-dessous :

- Il permet une personnalisation totale du cycle de vie du logiciel. *SLIM-Estimate* est fait de telle sorte qu'elle reflète l'unique nomenclature et les relations du processus de développement de logiciel d'une organisation quelconque.

- Il supporte séparément ou en combinaison, les méthodes les plus populaires d'estimation de la taille. La taille d'un système proposé peut être estimé par l'utilisation des techniques d'estimation, les points de fonction, la mesure de la taille de l'interface graphique de l'utilisateur (objet), et la mesure de la taille par module.

- Il saisit les données historiques. La totalité des sous-ensembles ou ceux sélectionnés des données historiques de l'organisation concernée peuvent être enregistrées dans *SLIM-Estimate.*

- Il fait une mise à jour continue du graphique visualisé du plan de projet. Les paramètres ayant un impact sur le plan du projet peuvent être changés graphiquement et l'effet résultant sur le programme, l'effort, le coût, la qualité et le risque peut être observé en temps réel.

- Il prévoit la fiabilité. L'effet du changement du personnel, du coût et du programme sur la fiabilité du produit peut être évalué (le taux de défectuosité et le temps moyen du défaut à la livraison).

- Il évalue le risque. L'effet du changement du personnel, du coût et du programme sur les probabilités de respecter les échéances, le budget et les niveaux de qualité du projet peut être évalué.

- Il sauvegarde de multiples plans et scénarios; il permet de recharger les plans pour des fins d'analyses additionnelles. Dans cet outil, «*Solution Log* » peut être utilisé pour la gestion de la configuration du plan ou simplement pour comparer les différents scénarios. Plus de dix plans peuvent être enregistrés, mais aucun ne peut être rétabli immédiatement.

- Il valide les buts c'est-à-dire qu'il facilite la comparaison graphique des scénarios planifiés aussi bien avec la performance historique qu'avec l'industrie.

- Il crée des présentations personnalisées contenant une multitude de tables et de graphiques qui peuvent être sélectionnés.

- Il supporte un processus exhaustif de planification.

- Il intègre d'autres logiciels basés sur Windows de Microsoft.

- Il contient plus de vingt ans d'expérience dans l'industrie. Les équations fondamentales de *SLIM-Estimate* dépendent d'une base de données de cinq milles projets et plus, et ces données sont accessibles à l'utilisateur.

SLIM_Metrics : c'est la version basée sur Windows du fameux répertoire des métriques et mesures PADS (« *Productivity Analysis Database System »).* La partie de l'entrée de donnée SLIM-Metrics est complète et peut être personnalisée. Sa partie d'analyse contient des outils statistiques, et graphiques, des outils de requêtes et de rapports nécessaires à l'évaluation et la comparaison de la performance des projets. Il est utilisé pour déterminer la performance, la position concurrentielle, et les tendances du développement.

- Il saisit les données historiques, c'est-à-dire que toutes les données historiques peuvent être enregistrées dans une base de donnée relationnelle ouverte.

- Il supporte une entrée de donnée efficace et logique. SLIM profite des caractéristiques de la boîte de dialogue de Windows pour faire une entrée de données rapide, logique et sans erreur.

- Il contient une capacité puissante de requêtes : des sous-ensembles spécifiques de données peuvent être créés et comparés entre eux.

- Il supporte de multiples écrans, c'est à dire qu'il supporte la création d'un nombre illimité de graphiques et d'écrans tabulaires à l'intérieur d'une seule feuille de travail.

- Il compare plus de cinq ensembles de données sur un même graphique.

- Il permet une identification immédiate d'un projet sur n'importe quel graphique avec la souris.

- Il représente un projet spécial sur différents graphiques, ce qui est utile pour la visualisation des compromis et des relations de cause à effet.

- Il contient de nombreuses caractéristiques qui permettent de gagner du temps.

- Il inclut les *benchmark* de l'industrie. *SLIM-Métrics* est accompagné de lignes de tendances de chaque mesure et métrique de l'industrie mises à jour annuellement.

- Il contient des outils pour une analyse statistique et de régression incluant quatre différentes courbes appropriées aux algorithmes.

- Il représente une base de données qui peut être personnifiée ou personnalisée.

- Il produit des sorties de qualité, c'est-à-dire de bonnes impressions des graphiques.

- Il supporte l'intégration complète de Windows, c'est à dire que les données, les graphiques et les tables, peuvent être déplacés de part et d'autre des produits de Windows.

SLIM-Control : il est utilisé pour s'assurer que les projets rencontrent les attentes et pour faire des corrections tactiques en plein projet.

- Il fournit une évaluation du projet à travers trois couleurs : le vert signifie que le projet est conforme aux objectifs, le jaune avise que le projet doit être surveillé de près et le rouge indique qu'il nécessite une action.

- Il utilise des techniques de contrôle du processus statistique.

- Il supporte l'usage des métriques.

- Il contient des écrans d'analyse qui peuvent être configurés pour inclure les graphiques les plus significatifs utiles à la rationalisation de l'analyse.

- Il fournit une prévision crédible à l'exécution. Les prévisions sont basées sur ce qui a été effectué à temps. Ainsi, en utilisant la courbe appropriée à la technique, SLIM-

Control trouve le programme, le coût et la fiabilité les plus probables basés sur toutes les métriques saisies

- Il simule différents scénarios « *what if* ».
- Il supporte la validation des plans et des prévisions, c'est à dire qu'il permet de comparer les prévisions et les résultats actuels à la performance historique. Il établit ainsi, la crédibilité des prévisions et facilite l'identification des attentes irréalistes.
- Il permet la création de présentations personnalisées contenant de multiples tables et graphiques que l'on peut sélectionner.
- Il inclut un répertoire de solution.
- Il nécessite un minimum d'entrées de données pour produire un maximum d'impact.
- Il permet une personnalisation totale du logiciel de cycle de vie.
- Il s'intègre aux autres logiciels basés sur Windows de Microsoft.
- Il est conforme aux rapports mensuel et hebdomadaire (une fois par semaine).

Avec leurs différentes caractéristiques, ces trois outils revêtent des avantages qui font de SLIM un modèle intéressant. Cependant, toute cette panoplie d'atouts suffit-elle pour juger de la fiabilité de ce progiciel d'estimation du coût logiciel? Fournir une réponse à cette question serait l'équivalent d'évaluer la qualité des estimations faites avec SLIM et de bien les analyser; ce qui définit l'objectif principal de cette recherche.

CHAPITRE III

MÉTHODOLOGIE : COMMENT SERA FAITE LA MESURE DE LA QUALITÉ DES ESTIMATIONS DU PROGICIEL SLIM

3.1 RAPPELS

L'estimation est une activité fondamentale dans la planification d'un projet. Les estimations initiales du coût et de la durée du projet sont les éléments primaires dans une prise de décision que le projet soit poursuivi ou pas. Une fois que le projet est financé, ces estimations, et leurs subséquentes forment la base de toute la planification à suivre au niveau du personnel, de l'équipement, des outils, etc. Malheureusement, dans le domaine du développement de logiciel, le processus d'estimation est souvent complètement ignoré, exécuté de façon médiocre (en utilisant des méthodes et des données inadéquates), ou ses résultats sont mal interprétés ou mal utilisés[5].

Cependant, le problème fondamental de l'estimation du logiciel, repose sur le fait que dans le but de financer le projet, le coût et la durée de développement doivent être connus. Plusieurs organisations s'attendent à ce que les estimations budgétaires aient une précision de +/- 10%, avant même la définition des spécifications. Malheureusement pour les gestionnaires de logiciel, ce degré de précision espéré à l'état primitif du projet n'est pas théoriquement possible[6]. À l'étape initiale du concept du produit, l'effort (coût) peut varier de 0.25 à 4.0 fois l'effort estimé, pendant que la durée varie de 0.60 à 1.60 fois la durée estimée (Boehm et al., 1995). Le manque de précision inhérent aux estimations primitives entraînent l'insatisfaction de la clientèle, la perte de confiance, et le dépassement du coût et de la durée prévus, même quand le coût et la durée actuels s'avèrent être à l'intérieur de la marge d'erreur des estimations originales. Afin de pallier à cela, les estimations doivent être de qualité; ainsi, la fiabilité du modèle utilisé pour faire l'estimation doit être étudiée. Dans le cas, de cette

---

5 Source : http://www.logikos.com/projmgmt/ProjEst1.html

6 Source : Idem

recherche, il s'agit du progiciel SLIM qui sera sujet à test sur la base de données ISBSG pour vérifier s'il rencontre l'approche classique du génie logiciel en vertu de laquelle *un modèle de productivité sera considéré comme « bon » s'il est capable de rencontrer le critère d'erreur relative moyenne de +/- 25% pour 75% des observations* (Abran, et Robillard, 1993). Ce critère d'évaluation est celui recommandé par plusieurs auteurs spécialisés dans ce domaine du génie logiciel (Conte, 1986; Verner, 1992).

3.2 DESCRIPTION DE DONNÉES

Une analyse empirique crédible doit être basée sur un grand échantillon d'observations (Abran et al.,1997). Avoir accès à une collection d'observations de taille adéquate est souvent plus qu'une problématique. Et cela est également vrai pour un bon nombre de praticiens travaillant dans les environnements commerciaux. Ces praticiens se retrouvent toujours à recueillir leur propre échantillon historique qui n'est utilisable qu'à court terme, ou à payer de gros montant pour accéder à une expertise spécialisée basée sur la commercialisation des bases de données de marque déposée.

Une alternative est apparue récemment avec la disponibilité d'un entrepôt provenant de ISBSG (*International Software Benchmarking Standards Group)*. Ce groupe s'intéresse à la collecte, la validation et la publication d'un entrepôt de données historiques des projets de développement logiciel. En décembre 1999, ISBSG a publié la sixième version de son entrepôt qui contient des données historiques de 789 projets de développement de logiciel complété entre 1989 et 1998.

3.2.1 Analyse de l'échantillon de données

Ces projets ont été menés dans 20 pays dont, 35% en Asie-Pacifique, 34,4% en Amérique du nord, 0,4% en Amérique du sud, 29,2% en Europe et le 1% restant identifie 7 pays non spécifiés. La majorité des projets est conçue pour les organisations de type : administration publique (13,4% des projets), finance, services d'affaires et de propriété (12,4%), banque (14,8%), et assurance (13,1%). Les types de développement de projet sont répartis en trois parties principales dont le nouveau développement (53,4%), l'amélioration (40,7%), le redéveloppement (5,7%), et 0,2% correspondent à d'autres types de développement. Les deux principales catégories des projets sont le MIS (*management information systems*, 42,7%), et les systèmes de transaction et de production (33,3%). Un peu plus des trois quarts des projets (76,9%) sont développés sur place, pour une unité d'affaire interne. 30,4% des projets impliquent une architecture client-serveur, 61,7% une plate-forme « *mainframe* » et seulement 25,5% des projets sont de nature générique. Les langages de développement de troisième génération (3GL) sont utilisés dans 45,1% des projets et ceux de quatrième génération (4GL) sont utilisés dans 45,5% des projets.

3.2.2 Critères de base

Parmi les 789 projets de l'entrepôt de ISBSG-1999, seul ceux qui répondent à ces caractéristiques seront pris en considération (Abran et al., 1997):

- Il n'y a aucun doute sur la validité de la donnée; c'est-à-dire que la base de données ISBSG n'a pas marqué un projet ayant une donnée incertaine et qu'elle l'a retenue pour ses propres analyses.
- L'effort est connu.
- La durée est connue.
- Le langage utilisé pour la programmation est connu.
- L'effort est supérieure ou égale à 400 heures/personne.

Les quatre premiers critères sont faciles à comprendre. Cependant, le cinquième a été choisi sur la base que les efforts impliquant ceux de moins de 400 heures personne sont souvent considérés comme trop petits pour faire l'objet d'une structure officielle de projet dans plusieurs organisations selon l'expérience de Abran, Oligny et Bourque (1997) dans le domaine. Seuls 497 projets rencontrent ces critères et une analyse statistique de l'échantillon est décrite comme suit :

Tableau 3.1 : Analyse statistique descriptive de l'échantillon

	Durée (mois)	Effort (heures/personne)
Nombre d'observations	497	497
Valeur minimale	1	400
Valeur maximale	84	138883
Valeur moyenne	10,5	6949,08
Écart-type	9,2	13107,61
Médiane	8	2680

Selon le langage de programmation utilisé, ces 497 projets se répartissent comme suit :

Tableau 3.2: Disparité des différents langages de programmation

Langage de programmation	Nombre de projets	Langage de programmation	Nombre de projets
Access	17	Oracle	26
Assembler	2	Oracle Forms	2
C	15	Oracle SQL	2
C++	21	Other 3GL	3
Clipper	4	Other 4GL	31
Cobol	106	Other APG	16
Cobol II	21	Pascal	2
CSP	7	PL/I	29
Easytrieve	8	Powerbuilder	18
Focus	2	SQL	20
Ideal	6	SQL Forms	3
Ingres	1	SQL Windows	4
Java	4	Telon	23
Natural	41	Unix Script	1
2GL	1	Visual Basic	34
Inconnus	21		

Cependant parmi ces 497 projets, 21 sont dépourvus d'une identification des langages de programmation utilisés, et du facteur d'équivalence qui permet la conversion des lignes de codes en points de fonction, qui est l'unité principale de la taille des différents projets de ISBSG. Il s'agit des projets logiciels développés en Sheer, Rally, PL/SQL, HPS, Drift et Abap.

De plus, 6 autres projets s'avèrent impossibles à estimer avec le progiciel SLIM, car, après conversion en lignes de codes, leur taille est inférieure à 1000 qui représente le minimum qu'un projet doit avoir pour pouvoir être estimé par SLIM. En fait, pour que l'effort d'un projet soit estimé par SLIM, sa taille doit être comprise entre 1000 et 25000000 lignes de code. Ainsi, seuls 470 des 497 projets répondant aux critères sont disponibles pour mesurer la qualité des estimations faites avec SLIM.

3.3 MESURE DE LA QUALITÉ DES ESTIMATIONS FAITES AVEC SLIM

3.3.1 Modèle basé sur le coût unitaire moyen (heure/personne) : Analyse de l'erreur

Le but de cette recherche est de voir à quel point l'effort estimé par le progiciel SLIM (*Eest*) concorde avec l'effort réel ou actuel (*Eact*). Si les modèles étaient parfaits, alors pour tous les projets, *Eest* devrait égaler *Eact.* Et, cela est rarement le cas, si jamais égalité il y a (Kemerer, 1987). Selon lui, il y a plusieurs méthodes possibles d'évaluer les estimation de l'effort. Un approche simple d'analyse est de voir le différence entre *Eest* et *Eact*. Le problème avec cette erreur absolue est que l'importance de la taille de l'erreur varie suivant la taille du projet. En tenant compte de cela, Boehm (1981), a recommandé le test du pourcentage d'erreur aussi appelé erreur relative (*RE*) par Conte et al. en 1986 (El Emam, 1998). Elle est définie comme suit :

$$RE = \frac{Eact - Eest}{Eact}$$

Les erreurs peuvent être de deux types : des sous-estimations où *Eest* <*Eact* et des surestimations où c'est le contraire qui se présente. Ces deux erreurs ont de sérieux impacts sur les projets. De grandes sous-estimations entraîneront un manque de personnel pour le projet, et, comme les échéances approchent, le gestionnaire du projet aura tendance à ajouter du personnel au projet. Cela résulte en un phénomène appelé la loi de Brooks (1975) que voici : « ajouter du personnel à un projet logiciel en retard, ne le rend que plus tardif »

(Kitchenham, 1998). D'un autre côté, les surestimations peuvent aussi être coûteuses au personnel qui, notant le ralentissement du projet, devient moins productif. C'est la loi de Parkinson qui soutient que « le travail s'étend pour combler le temps disponible pour son achèvement ».

Compte tenu du problème sérieux de ces erreurs de sous-estimations et de surestimations, Conte et al. (1986) ont suggéré l'erreur relative majeure (*MRE*) qui est la valeur absolue de *RE*.

$$MRE = |RE| = \left| \frac{Eact - Eest}{Eact} \right|$$

Ainsi, plus la valeur de *MRE* est petite, meilleure est la prédiction, c'est à dire que l'estimation est plus précise et elle reflète moins d'erreur (Conte et al., 1986; El Emam, 1998). Pour un ensemble de *n* projets, la moyenne de l'erreur relative majeure (*MMRE*) peut être calculée par la formule suivante :

$$MMRE = \overline{MRE} = \frac{1}{n} \sum_{i=1}^{n} MREi$$

Où n représente le nombre total d'occurrences (projets logiciels) dans un ensemble particuliers de données, et i est égal à la ième occurrence de ce même ensemble de données. De même que le *MRE*, plus que le *MMRE* est faible, meilleur est le modèle d'estimation; et pour que le modèle soit acceptable, le *MMRE* doit être inférieur ou égal à 0.25 (Conte et al., 1986; El Emam, 1998).

En plus de cela, il y a la racine carrée de l'erreur moyenne (*RMS*), la racine carrée de l'erreur relative moyenne ($\overline{RMS}$)et le niveau de prédiction (*PRED(l)*). Tout d'abord, plus les $\overline{RMS}$

et *RSM* sont faibles, meilleure est l'aptitude du modèle à prévoir la performance actuelle. La racine carrée de l'erreur moyenne est déterminée par la formule suivante :

$$RMS = \left(\overline{SE}\right)^{1/2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Eact_i - Eest_i\right)^2}$$

Et l'équation permettant de calculer la racine carrée de l'erreur relative moyenne est définie comme suit :

$$\overline{RMS} = \frac{RMS}{\frac{1}{n}\sum_{i=1}^{n} Eact_i}$$

Pour ce qui est du niveau de prédiction, il reflète le pourcentage des estimations d'un projet, étant donné un ensemble de données qui se retrouvent à l'intérieur d'un écart de pourcentage prédéfini de leurs valeurs actuelles. Cette mesure est définie par (Conte et al., 1986; El Emam, 1998) :

$$PRED(l) = \frac{k}{n}$$

Où k équivaut au nombre de projets logiciels dans un ensemble particuliers de données de *n* projets dont *MRE =< k/n.*

Par exemple, si *PRED(0.25)* = 0.83, alors, 83% des valeurs prévues se retrouvent à l'intérieur de +/-25% de leurs valeurs actuelles (Conte et al., 1986, El Emam, 1998). Ainsi, les auteurs ont conclu qu'un critère acceptable pour un modèle d'estimation de l'effort est *PRED*(0.25)>=0.75.

3.3.2 Modèle basé sur la droite de régression linéaire

Il s'agit là de mesurer la corrélation entre l'effort estimé et l'effort actuel. La régression sera faite en utilisant l'effort actuel comme variable dépendante et l'effort estimé comme variable explicative (Kemerer, 1987, Abran et Robillard, 1993).

En résumé, deux tests seront utilisés pour évaluer la qualité des estimations du progiciel SLIM (comme recommandé par Theabaut, 1986) : l'analyse de l'erreur et la régression linéaire. Ces tests n'ont pas seulement l'avantage d'être acceptés dans la littérature, mais aussi, ils ont été proposés par certains développeurs des modèles (Kemerer, 1987).

À cela suivra une analyse et une interprétation des résultats trouvés. Ces deux dernières phases consistent à évaluer et expliquer la relation entre la taille fonctionnelle d'un logiciel quelconque et l'effort de développement de ce même produit. Ce qui permettra de déduire si le modèle de productivité basé sur les lignes de code SLIM répond aux critères des « bons modèles » en génie logiciel, à savoir, *un modèle de productivité sera considéré comme « bon », s'il est capable de rencontrer le critère d'erreur relative moyenne de +/-25% pour 75% des observations* (Conte, 1986; Verner, 1992; Abran et Robillard, 1993).

BIBLIOGRAPHIE

Abdel-Hamid, T. et S. E. Madnick 1991. *Software Project Dynamics and Integrated Approach.* Englewood Cliffs, NJ, Prentice Hall.

Abran, A. et P. N. Robillard 1993. « Analyse comparative de la fiabilité des points de fonction comme modèle de productivité», *Revue de Liaison de la recherche en informatique cognitive des organisations [ICO]* , Vol. 4, no 3-4, p. 16-24.

Albrecht, A.J., J.Gaffney 1983, « Software function, source lines of code, and development effort prediction » SE Vol. 9, no 6, November, p 639-648.

Basili, V.R. 1980, *Model and metrics for software management and engineering*, IEEE Computer Society Press.

Boehm, B. W. 1981. *Software engineering economics*. Englewood Cliff, New Jersey, Prentice-Hall Inc.

Boehm, B. W. 1984. "Software Engineering Economics." *IEEE Transaction on Software Engineering*, SE Vol. 10, no 1, p. 4-21.

Boehm, B., B. Clark et al. 1995. « Cost Models for Future Software Life Cycle Processes: COCOMO 2.0 ». *Annals of Software Engineering Special Volume on Software Process and Product Measurement*. J. D. Arthur and S. M. Henry. Amsterdam, The Netherlands, Science Publishers.

Bourque, P. 1988. « Développement d'un modèle statistique d'estimation du nombre de lignes de code fondé sur des métriques de spécification », Faculté des Sciences - Département

de mathématiques et d'informatique, Sherbrooke, Québec, Université de Sherbrooke, 139 p.

Briand, L. C., K. El Emam, et al. 1998. *An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques*. Germany, Fraunhofer Institute for Experimental Software Engineering.

Brooks, F.B. 1975. *The mythical manmonth. Essays on software engineering*, Addison-Wesley.

Conte, S.D., H.E. Dunsmore, V.Y. Shen 1986. *Software engineering metrics and models*. Menlo Park : The Benjamin/Cummings Publishing Company, Inc.

Côté, V., P. Bourque et al. 1988. «Software Metrics: An Overview of Recent Results ». reproduced in *De Grace, P., Hulet Stahl, L., 'The Olduvai Imperative-CASE and the State of Software Engineering Practice, Yourdon Press Computing Series, Prentice Hall, 1993. Originally in Journal of Systems and Software,* vol. 8, no 2, p. 121-131.

DeMarco, T. 1982. *Controlling software projects : management, measurement and estimation,* Yourdon Press, New York.

DeMarco, T. 1984 « An algorithm for sizing software products », *Performance Evaluation review*, Vol. 12, p. 13-22.

Dion, F. et A. Abran 1999. « Decisions and Justifications in the Context of Industrial-Level Software Engineering ». *International Workshop on Software Measurement (IWSM)*, Lac Supérieur, Québec, p. 2-9.

« Envision a World », *QSM associates,* http://www.qsma.com.

Frieman, F.R. et R.D. Park 1979. « PRICE software model-version3 : An overview », In *proceedings IEEE PINY workshop on quantitative software models,* p. 32-41.

Gaffney, J.E., R.D Cruikshank 1997. «How to estimate software system size », *Software Engineering Project Management,* 531 p.

Garmus, D. et D. Herron 1996. « Effective Early Estimation ». *Software development*, July.p. 57-65.

Heemstra, F.J. 1989. *How expensive is software? Estimation an control of software-development*, Kluwer.

Heemstra, F.J. 1992. « Software cost estimation », *Information and Software Technology*, Guildford, October, p. 627-639.

Herd, J.R., J. Postak, W. Russel, K. Steward 1977. « Software cost estimation study- Study Results », Final technical report RADC-TR-77-220, Vol. 1, Doty Assoc., Rockville, Maryland.

Ingram, T. 1994. « Managing client/server and open systems projectsL a 10 years study of 62 Mission-Critical Projets » *Project management journal*, Juin.

ISBSG - Worldwide Software Development - The Benchmark. Victoria, Australia International Software Benchmarking Standards Group, 1999.

Jones, C. 1986. *Programming productivity,* McGraw-Hill.

Jones, C. 1994. *Assessment and Control of Software Risks*. Englewood Cliffs, NJ, Yourdon Press.

Jones, C. 1997. Applied Software Measurement, Assuring Productivity and Quality. New York, NY, McGraw Hill.

Kemerer, C. F. 1987. « An empirical validation of software cost estimation models », *Communications of the ACM*, Vol. 30, no 5, Mai.

Kitchenham, B. 1998. « The Certainty of Uncertainty ». *FESMA 98 - Business Improvement Through Software Measurement*, Antwerpen, Belgium.

Kitchenham, B.A et N.R. Taylor, N.R . 1984. « Software cost models », *ICL technical journal*, Vol. 4, no 1, May, p. 73-102.

Lederer, A. L. et J. Prasad 1993. « Systems development and cost estimating. Challenges and guidelines ». *Information Systems Management* Vol. 10, no 4, p. 37-41.

McIntyre, J. « Project estimating :Effecive Sofware Estimating », Telling and selling the Software Estimating Story, Logikos Inc., part I, http://www.logikos.com/projmgmt/ProjEst1.html.

Oligny, S., P. Bourque et A. Abran 1997. « An emprical assesment of projet duration models in software engineering ». *The eight European Software Control an Metrics Conference (ESCOM'97)*, Berlin Germany, May.

Oligny, S., P. Bourque, et al. 1997. « An Empirical Assessment of Project Duration Models in Software Engineering ». *8th European Software Control and Metrics Conference (ESCOM'97)*, Berlin, Germany, European Software Control and Metrics Conference, May.

Putnam, L.H. 1978. « A general empirical solution to the macro software sizing and estimating problem », *IEEE Transactions on Software*, SE Vol. 4, no 4, p 345-361.

Rubin, H.A. 1983. « Macro-estimation of software development parameters : The ESTIMACS system », *IEEE SOFTAIR Conference on software development tools, Techniques and Alternatives.*

Stroian, V. 1999 « Un environnement automatisé pour un processus transparent d'estimation fondé sur la base de données du International Software Benchmarking Standards Group (ISBSG) », Département d'informatique, Montréal, Université du Québec à Montréal, 47 p.

Stutzke, R.D. 1997. « Software estimating technology :a survey », *Software engineering Project Management*, p. 218-229

The Standish Group. 1995. « CHAOS ». http://www.standishgroup.com/chaos.html.

Theabaut, S.M. 1986. « Model evaluation in software metrics research », *Computer Science and Statistics Proceedings of the 15th Symposium on the Iinterface*, Houston, Texas, p. 277-285

Van Genuchten, M. et H. Koolen 1991. « On the Use of Software Cost Models ». *Information & Management [IFM]* Vol.21, no 1 p. 37-44.

Van Genuchten, M. et. Koolen 1991. « On the Use of Software Cost Models. » *Information & Management [IFM]*, Vol. 21, no1, p. 37-44.

Verner, J. et T. Graham 1992 « A Software size Model », *IEEE Transactions on software enginerring*, Vol. 18, no 4, Avril, p.149-158

Vicinanza, ST., M.J. Prietula et T. Mukhopadyay 1990. « Case-based reasoning in software effort estimation », *Proc. 11th International conference on information systems*, Cophenhagen, Denmark, 16-19 December, p. 149-158

Walkerken, F. et. Jeffery 1996. « Software cost estimation » *A review of models, process and practice*. Sydney, Australia, Centre for Advanced Empirical Software Research, School of Information Systems, University of New South Wales.

Wolverton, R .W. 1974. « The cost of developing large-scale software », *IEEE Transaction on Computer.*

APPENDICE A

CADRE DE BASILI

<table>
<tr><td colspan="6">Définition</td></tr>
<tr><td>Motivation</td><td>Objet: sujet</td><td>Objectifs</td><td>Utilisateurs</td><td>Domaine</td><td>Étendue</td></tr>
<tr><td>Améliorer la qualité de la prise de décision d’estimation des gestionnaires de projets informatiques</td><td>Le modèle d’estimation SLIM</td><td>Évaluer la qualité des estimations et bien les analyser</td><td>• Gestionnaires en informatique (chefs de projet, décideurs)
• Praticiens qui estiment les coûts en développement de logiciels</td><td>MIS: développement de logiciel : Modèle de productivité à posteriori : modèles d’estimation des coûts logiciels</td><td>Base de données ISBSG</td></tr>
<tr><td colspan="6">Planification</td></tr>
<tr><td colspan="2">Conception</td><td colspan="2">Critères</td><td colspan="2">Sélection de mesure</td></tr>
<tr><td colspan="2">• Présentation du domaine de l’estimation coût-logiciel
• Se familiariser avec SLIM
• Présentation du modèle de productivité SLIM
• Analyse de l’échantillon
• Évaluation de la qualité de l’estimation de SLIM avec la base de données ISBSG</td><td colspan="2">Critères directs :
• Analyse d’erreur (*)
*Différence entre effort estimé et effort actuel
*Pourcentage d’erreur (est-act)/act
*Erreur relative moyenne (valeur absolue du pourcentage d’erreur
• Régression pour mesurer la corrélation entre l’effort estimé et l’effort actuel (effort = variable dépendante)

Critères indirects :
Critères de base de ISBSG-1999 (échantillon) :
• Aucun doute sur la validité de la donnée; c’est à dire que la base de données ISBSG n’a pas marqué un projet ayant une donnée incertaine et qu’elle l’a retenu pour ses propres analyses.
• L’effort est connu
• La durée est connue
• Le langage est connu
• Effort >= 400 heures/personnes</td><td colspan="2">• Modèle basé sur le coût unitaire moyen (jour/personne) de l’ensemble des observations de l’échantillon (facteur de productivité)

• Modèle basé sur la droite de régression linéaire (effort = variable dépendante)</td></tr>
</table>

Exécution		
Préparation	Déroulement	Analyse de données
• Présentation du domaine de l'estimation coût-logiciel • Se familiariser avec SLIM • Présentation du modèle de productivité SLIM • Analyse de l'échantillon • Évaluation de la qualité de l'estimation de SLIM avec la base de données ISBSG	• Collecte de documents sur le sujet • Tests avec des projets de ISBSG • Collecte d'informations sur le progiciel • Étude de la base de données ISBSG • Épuration des données et de ISBSG et entrée de données épurée dans SLIM-estimate	• Revue de littérature • Comparaison des efforts réels et estimés • Résumé des principales caractéristiques du progiciel SLIM • Méthodologie
Interprétation		
Contexte d'interprétation	Extrapolation	Travaux futurs