UNIVERSITÉ DU QUÉBEC À MONTRÉAL

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

MÉMOIRE

PRÉSENTÉ COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE DE GESTION

PAR

IPHIGÉNIE NDIAYE

Avril 2001

Mémoire approuvé par :

Alain Abran,
Professeur au Département d'informatique
Directeur au Laboratoire de recherche en gestion des logiciels

Ghislain Lévesque
Professeur au département d'informatique
Directeur au programme du doctorat en informatique cognitive

REMERCIEMENTS

Tout d'abord, je remercie le Seigneur Tout-Puissant de m'avoir accordée la force et le courage sans quoi je n'aurais pu réaliser ce projet.

Ensuite, un remerciement spécial au Dr Alain Abran et au Dr Ghislain Lévesque, mes directeurs pour leur support, leur compétence et leurs conseils pertinents. Cette recherche est de beaucoup le résultat direct de leur disponibilité, même à distance, de leurs remarques et de leurs suggestions réfléchies. Je leur en suis très reconnaissante et leur exprime toute ma gratitude.

Je témoigne ma profonde gratitude à toute l'équipe du Laboratoire de recherche en gestion des logiciels de l'UQAM, tout particulièrement à Michèle Hébert et François Cossette pour leur support et leur collaboration. Qu'aurai-je fait sans vous deux?

Un gros merci à M. Bertrand Fournier pour ses conseils pratiques dans l'analyse statistique de mes résultats, à Julie Hudon et Monique Bissonnette du Département d'informatique pour leur assistance.

Je tiens à remercier également Mme Michèle Picard de CGI Telecom pour avoir parrainer mon projet lors du concours Claude Laporte.

J'adresse un particulier et chaleureux remerciement à mes parents Pierre et Agnès, pour leur amour, leur soutien et leurs encouragements. Sans eux, tout ceci serait impossible.

Finalement, je remercie toutes les personnes qui de près ou de loin m'ont apporté leur support moral tout au long de ma maîtrise, en particulier : Évelyne, Delphine, Marie Odile, Marie-Thérèse, Gabriel, Marie-Angélique, Abdoulaye, Fatou, Séni, Carmen, Adja, Marie-Jeanne, Étienne, Solange, Carole, Valery, Anne-Nathalie, Kajus. Je ne peux passer sous silence votre appui.
Un gros gros MERCI !!!

À Papa et Maman en guise de reconnaissance, je vous dédie ce mémoire. Merci pour tout!
Je le dédie également à mes frères et sœurs Yolande, Joseph Hervé, Marthe, Simone et Clément. Qu'il soit pour vous source d'inspiration et un exemple de patience et de persévérance dans toutes vos entreprises futures.

Avec tout mon amour,

TABLE DES MATIÈRES

LISTE DES ACRONYMES

ISBSG	International Software Benchmarking Standard Group
MMRE	Mean magnitude of relative error – Moyenne de l'erreur relative moyenne
MRE	Magnitude of relative error – erreur relative moyenne
PF	Points de fonction
PHR	Hour per person – Heure par personne
RE	Relative error – Erreur relative
RMS	Root mean square error – Racine carrée de l'erreur moyenne
$\overline{\text{RMS}}$	Relative root mean square error – Racine carrée de l'erreur relative moyenne
SLIM	Software Life-Cycle Model

ABSTRACT

Managers of software development initiatives must routinely make crucial decisions in contexts where there are many unknowns regarding the expected outcomes, for example when making project estimations for both effort and duration of software development. And it often happens that software projects are more expensive than estimated and with late completion. Many of these serious consequences are outcome of badly informed decisions earlier on in the development process. So, it is important to obtain good estimates early in software project life, and to understand the potential range of variations of such estimates. To support manager in the estimation, there are, many parametrics estimation models which are proposed in the literature and estimation software tools in the market place; however, very little in known about the quality of the estimates of such models, including about the SLIM model (Software Life-Cycle Model), one such estimation tool available in the market place.

This paper presents an exploratory research which main purpose is to evaluate the quality of the estimates produced by the SLIM software estimation tool (Putnam's model, 1978). So, the results of this research will be useful to any manager in computer science and to any practitioner who have to estimate software costs development.

This study had been developed in three phases. In the first one, the projects in the repository of the International Software Benchmarking Standard Group are studied and data samples were created based on the criteria of the programming language types existing in ISBSG database.

In the second phase, the size and duration of each project are used as entry parameters in the SLIM tool to estimate each project's effort. Finally, in the last part of this study, this SLIM estimated effort is compared to the one already estimated by the automated environment of ISBSG database, which had been developed by Stroian in 1999 at the Software Engineering Management Research Laboratory. Results of this comparison show the differences between SLIM's estimation and the real effort of development. To verify these results, estimated effort and real effort have been correlated. In summary, the results indicate that SLIM does not respond to the criteria of «good models» in software engineering, that is, *a productivity model will be considered as «good», if it is able to meet the criteria of the mean relative* (Conte, 1986; Verner, 1992, Abran et Robillard, 1993).

KEY WORDS: ESTIMATION, MEASUREMENT, SLIM, ISBSG, SOFTWARE PROJECTS

ales dans un contexte où il y a beaucoup d'inconnus et d'incertitudes, ce qui fait du développement de logiciel une activité importante et complexe. La difficulté de celle-ci repose sur plusieurs autres activités telle que l'estimation de l'effort et de la développement du projet. Il arrive souvent que le logiciel soit plus ou moins cher que ce qui a été estimé, et qu'il soit livré avant ou après la date prévue; et dans ce cas, il s'avère également qu'il ne réponde pas aux demandes du client, ce qu mauvaise prise de décision. Ainsi, il est important d'estimer avec précision l'effort et la durée de développement du projet logiciel. Et pour cela, plusieurs modèles paramétriques sont e parmi lesquels se retrouve le progiciel SLIM (*Software life-cycle model*).

Ce texte qui suit présente une étude exploratoire dont l'objectif est de mesurer la fiabilité des estimations de SLIM (modèle de Putnam, 1978) afin d'améliorer la qualité des prises de décision des gestionnaires de projets logiciels. Les résultats seront donc utiles à tout gestionnaire en informatique et à tout praticien qui a à estimer les coûts de développement de logiciels.

Ce projet s'est déroulé en trois phases. Dans la première, les projets de ISBSG (International Software Benchmarking Standard Group) utilisés pour les fins de cette recherche sont étudiés. Ce qui a permis de construire divers échantillons de données sur les critères des différents types de langages de programmation existant dans la base de données ISBSG.

Dans la deuxième phase, la taille et la durée de chaque projet de l'échantillon sont entrées comme paramètre dans le progiciel SLIM afin de faire une estimation de l'effort de chaque projet. Finalement dans la dernière phase du projet, cet effort estimé par SLIM est comparé à celui déjà estimé par l'environnement automatisé du laboratoire de l'analyse de la base de donnée ISBSG, qui avait été développé au laboratoire de recherche en gestion des logiciels par Stroain en 1999. Les résultats de cette comparaison ont montré qu'il y a de grands écarts entre les évaluations de SLIM et celles de ISBSG, et entre les estimations de SLIM et l'effort fourni réellement. D'autre part, afin de vérifier ces résultats, la c estimé et l'effort réel a été évalué. Ce qui a permis de déduire que SLIM ne répond pas aux critères des «bons modèles» en génie logiciel, à savoir,
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). Aussi, les résultats montrent que l'environnement automatisé de ISBSG est un meilleur estimateur que SLIM. Une explication possible à cela est qu'à l'origine, SLIM a été développé en utilisant les données des projets reliés au département de la défense américaine, incluant les systèmes à temps réels (Kemerer, 1987); et ISBSG ne contient que des projets

MOTS CLÉS : ESTIMATION, MESURE, SLIM, ISBSG, PROJETS LOGICIELS

INTRODUCTION

L'ordinateur est actuellement, en ce début de millénaire, crucial pour notre société, avec comme élément clé le logiciel qui joue un rôle important dans tous les systèmes. Et dans cette même société, il y a une demande considérable pour plus de fonctionnalités, plus d'interfaces qui sont d'usage plus facile, qui ont moins de défauts et un temps de réponse plus rapide. Les développeurs doivent ainsi s'efforcer d'atteindre ces objectifs et, en même temps, réduire les coûts et la durée de développement. Le SPI (Software Process Improvement) préconisé par le SEI (Software Engineering Institute) sert à réaliser ces objectifs; et la planification des ressources et l'échéancier du projet sont identifiés comme les deux processus clés dans le modèle CMM (Capability Maturity Model) du SEI (Stutzke, 1997).

L'estimation du coût du logiciel et celle de la durée sont des éléments importants de la planification des projets logiciels. Durant les années 90, les praticiens ont accordé une attention particulière à l'estimation afin de concevoir de nouvelles façons de construire un logiciel et de fournir des estimations de coût et de temps plus précises et plus fiables (Stutzke, 1997). Quoique ce thème ait fait l'objet de beaucoup de publications tant sur les modèles disponibles que sur le processus lui-même d'estimation, en 1993, Abran et Robillard ont noté qu'il y avait une rareté de publications de recherches expérimentales indépendantes portant sur la fiabilité tant des modèles que des processus d'estimation. Ces modèles d'évaluation sont plus d'une vingtaine (Côté, Bourque, Oligny et Rivard, 1988), mais pour des raisons d'ordre pratique, cette recherche porte uniquement sur le progiciel SLIM.

Le sujet de la recherche sera donc d'évaluer la qualité, la fiabilité des estimations faites par SLIM. Et afin d'y arriver, le cadre de Basili est utilisé pour définir le projet dans le premier chapitre.

Le deuxième chapitre expose la problématique à partir d'une revue de la littérature où le sujet de l'estimation est présenté en tant que tel, et où le modèle de productivité SLIM est décrit.

À cela suit le chapitre de la méthodologie. Cette partie, quant à elle, présente la stratégie qualité des estimations du progiciel SLIM.

Le quatrième chapitre décrit l'analyse de la distribution de l'échantillon des données utilisées

Finalement, les cinquième, sixième et septième chapitres, les plus importants, sont consa à l'analyse des résultats, c'est-à-dire la comparaison entre les efforts réels et ceux estimés. Ce qui permet de déduire une interprétation et une conclusion en ce qui concerne la fiabilité de SLIM.

Le document se termine par une conclusion générale, dans laquelle sont présentés la synthèse de la recherche et les travaux futurs qui pourraient en découler.

CHAPITRE I

DÉFINITION

Ce chapitre présente une définition de la recherche qui a été effectuée. La motivation, le sujet, l'objectif, le domaine, l'étendue de la recherche et les utilisateurs des résultats y sont présentés (Appendice A).

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 la rap
le génie logiciel. Des décisions critiques ayant un impact sur le succès du projet et même sur l'organisation entière doivent être prises rapidement, sur la base d'une information limitée, 'une information abondante qui est difficilement maniable (Dion et Abran, 1999).

Les gestionnaires du développement de logiciel doivent donc 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 SUJET

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

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. Cette base de données ne contient que des projets du domaine MIS (*Management information Systems*).

CHAPITRE II

PROBLÉMATIQUE

Ce chapitre présente en premier le domaine de l'estimation du coût du logiciel. Ensuite, un aperçu est fait des préalables à l'évaluation du coût du logiciel ainsi que de certains modèles utilisés à cette fin, dont celui de Putnam (1978). Puis, ce modèle SLIM sera exposé par la suite.

D'après les résultats de plusieurs recherches (Heemstra, 1992), les projets logiciel ne sont pas sous contrôle et l'effort fourni lors du développement dépasse l'effort estimé, résultant soit en une livraison tardive du logiciel, soit en une livraison partielle des fonctionnalités attendues. Il n'y a aucun doute que l'estimation du logiciel est un problème sérieux 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 logiciels sont dis
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 COÛTS DES LOGICIELS (Heemstra, 1992)

La plupart des modèles actuels sont des modèles à deux étapes comme le décrit la figure 2.1. La première est la mesure de la taille et la deuxième fournit un facteur d'ajustement à l'estimation nominale de l'effort.

taille/le volume du logiciel à développer, exprimée en 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.

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

Vu que cet effort nominal ne tient pas compte des caractéristiques spécifiques connues du produit logiciel, de la façon dont ce dernier sera développé et des moyens de production, 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 application de ce facteur d'ajustement fournit une estimation plus

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 l'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 aintenir) 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 doit 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 souvent nécessaires avant que le produit soit défini de façon précise. La fonctionnalité du logiciel est alors difficile à définir de façon détaillée, 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

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 ode actuel dans un produit. Le code doit être localisé,

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

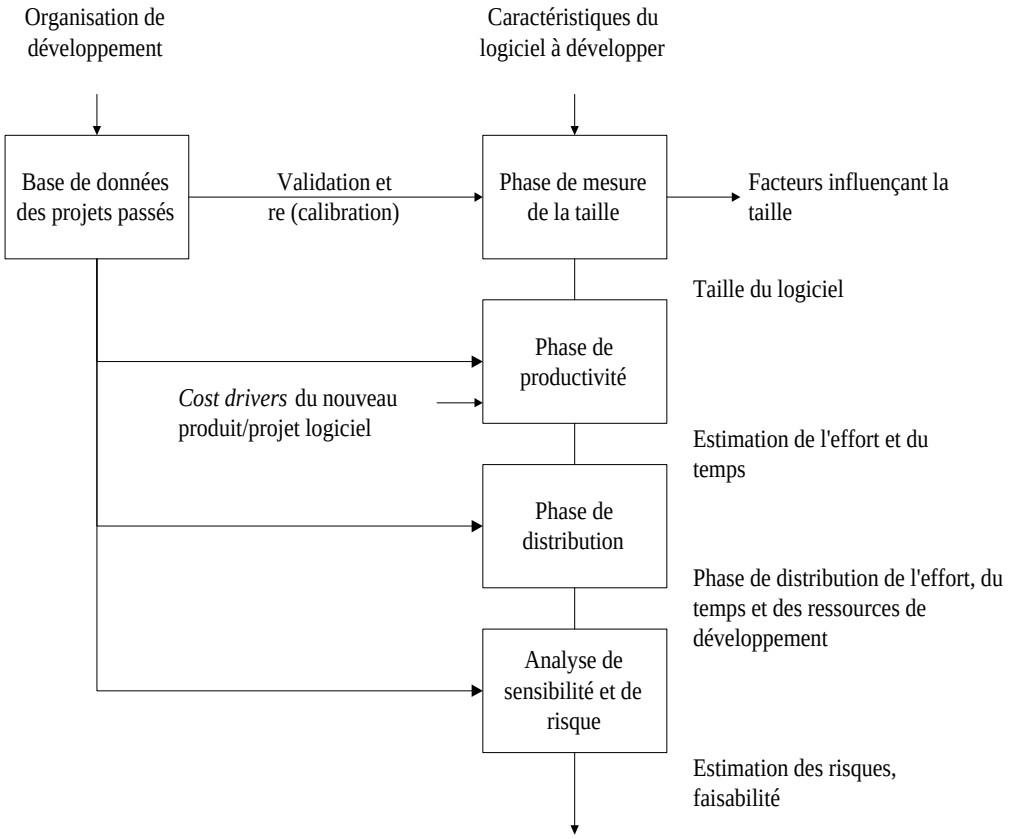


Figure 2.2 : Structure générale de l'estimation du coût[2]

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 les projets antérieurs 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, par exemple, 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

d'effort appliquée au développement du logiciel dans un court intervalle de temps n'est pas faisable (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).

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% des projets sont achevés à temps et ont respecté le budget initial; ce qui représente moins d'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).

---

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

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 ’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 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.

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:

- 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, début du projet. Des changements, adaptations et ajouts sont plus une règle qu'une exception et, conséquemment, les planifications et les budgets doivent être ajustés en cours de projet.

- Les caractéristiques et le développement du logiciel rendent l'estimation difficile; par exemple, le niveau d'abstraction, de complexité, 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

-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

En ce qui concerne les techniques d'estimation, il existe plusieurs classifications dans la littérature (Boehm, 1981; Basili, 1980; Van Genuchten et Koolen, 1991). La classification la plus citée est celle de 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 arités 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 ensuite 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 ensuite additionnées pour donner un coût total.

Bien que cela soit une façon de distinguer les diverses méthodes d'estimation, il est important 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 e d'estimation de coût le plus théorique qui doit être validé contre les données réelles et calibrées pour des environnements particuliers (Boehm, 1981; Walkerken et Jeffery, 1996; Stutzke, 1997).

Aucune technique particulière d'estimation de coût ne peut parer au manque 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 enregistre 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 2.1 (Boehm, 1981; Van Genuchten et Koolen, 1991; El Emam et al., 1998) :

Tableau 2.0:I : 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	• Peut ignorer les coûts du

[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.

	• Plus stable due à l'annulation de l'effet de collectivité • Stimule l'engagement individuel	« *system-level* » • Nécessite plus d'effort

Boehm (1981) montre que les forces et les 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 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 incluant COCOMO II (Boehm et al., 2000), Doty (Herd et al., 1977), SLIM (Putnam, 1978), PRICE-S (Frieman et Park, 1979), ESTIMACS (Rubin, 1983) et les points de fonction (Albrecht et Gaffney, 1983).

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

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

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

nt qui représente le coût.

ppement 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.*

[4] http://www.qsma.com, site de QSM associates

- 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

- Il prévoit la fiabilité du produit. 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

- 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 mille projets et plus, et il est prétendu que ces données sont accessibles à l'utilisateur, mais

SLIM-Metrics : c'est la version basée sur Windows du répertoire des 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 à 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 -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 étalonnages de l'industrie. *SLIM-Metrics* est accompagné de lignes de tendances de chaque mesure 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 cours de 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 mesures.
- 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 mesures 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 solutions.
- 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 mensuels et hebdomadaires (une fois par semaine).

Avec leurs différentes caractéristiques, ces trois outils revêtent des avantages qui font de SLIM un outil de gestion 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

Dans le chapitre précédent l'estimation du coût d'un logiciel a été vue dans son ensemble, justifiant ainsi l'objectif de cette recherche. Dans le présent chapitre les divers éléments de la méthodologie de cette recherche sont établis.

En effet, dans un premier temps, il sera fait un rappel des aspects fondamentaux de l'estimation du coût d'un logiciel, pour ensuite présenter la base de données ISBSG qui sera utilisée pour mesurer la qualité des estimations faites avec le progiciel SLIM. Et pour cette dernière deux sélections de mesure seront utilisées à savoir, l'analyse de l'erreur et la régression linéaire.

3.1 RAPPELS

L'estimation est une activité fondamentale dans la planification d'un projet. Les estimations 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 les 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 exécuté de façon médiocre (en utilisant des méthodes et des données inadéquates), ou ses résultats sont mal interpr [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

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

iciel. En décembre 1999, ISBSG a publié la sixième version de sa base de données qui contient des données historiques de 789 projets de développement de logiciel

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
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, seuls 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 comme 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 supérieur ou égal à 400 heures/personne.

Les quatre premiers critères sont faciles à comprendre. Cependant, le cinquième a ét
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
:

Tableau 3.2 : Analyse statistique descriptive de l'échantillon

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

s projets logiciels développés en Sheer, Rally, PL/SQL, HPS, Drift et Abap.

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 (Briand, 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 quand les échéances approchent, le gestionnaire du projet aura tendance à ajouter du personnel au projet. Cela résulterait 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 » .

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.

MRE, plus 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 actuelles. Cette mesure est définie par (Conte et al., 1986; El Emam, 1998) :

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

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).

:

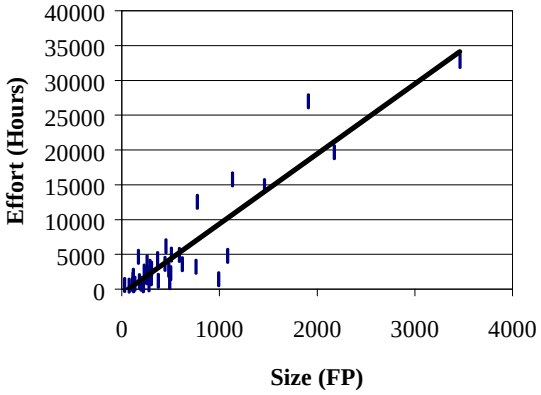


Figure 4.3 : Langage *Natural* dans son ensemble

Cette figure est représentée par l'équation de la droite suivante : Y = 10.053X – 648.91 avec un coefficient de corrélation R^2 = 0.8577.

-ensembles combinés. Il est alors jugé adéquat de développer un premier modèle avec un échantillon de données où les projets de taille 800 points de fonction feront l'objet d'un deuxième modèle. Ce qui est représenté par les figures 4.4 et 4.5 suivantes :

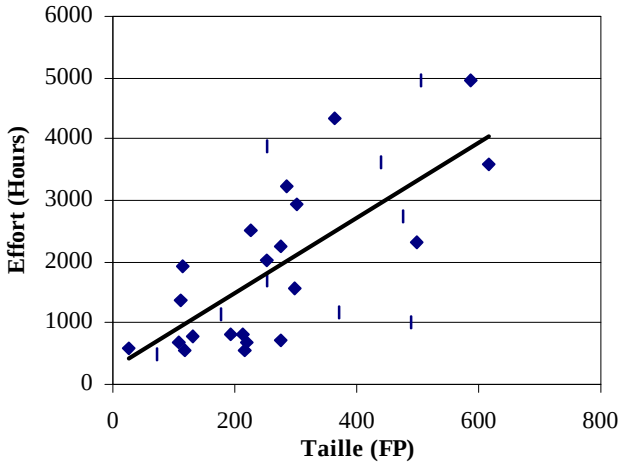


Figure 4.4 : Langage *Natural* et ses projets de taille inférieures à 620 points de fonction

Cette figure 4.4 représente précisément les projets de *Natural* dont la taille est comprise entre zéro et 620 points de fonction. En comparaison, cet échantillon montre une assez grande corrélation entre la taille des projets et l'effort de développement ($Y = 6.1355X + 264.98$ et $R^2 = 0.475$). Ainsi, cet échantillon peut être considéré approprié pour les tests de cette étude en ce qui concerne les applications développées dans le langage *Natural*. Cependant, pour le deuxième modèle de *Natural* qui comporte les projets de taille supérieure à 620 points de fonction, la figure 4.5 montre que la pente de la régression est différente de celle de la figure 4.4. De plus, il y a un déplacement majeur du point de rencontre de l'axe à zéro (la taille). La distribution reste quand même très bonne avec un équation de droite de $Y = 10.539X$ 1404.9 et un coefficient de corrélation de 74.24%.

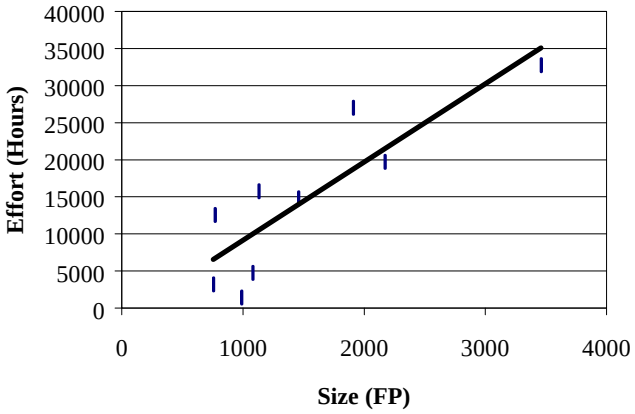


Figure 4.5 : Langage *Natural* et ses projets de taille supérieures à 620 points de fonction

Le même processus a été utilisé pour tous les autres langages des 470 projets. De plus, il faut rappeler qu'un échantillon avec seulement deux données n'est pas représentatif pour un ensemble de données. Évidemment par deux points ne passe qu'une seule droite, ce qui justifie le fait que le coefficient de corrélation entre l'effort et la taille du projet soit égal à un. Donc, ces échantillons sont trop petits pour des analyses; et cela explique l'élimination de certains langages qui ne contiennent que deux projets comme *Focus, Pascal, Assembler, Oracle Forms, Oracle SQL, Ingres, 2GL, Unix Script*. D'où la réduction du nombre de projets à 457 répartis comme suit :

Tableau 4.4 : Liste des échantillons (avec leurs points extrêmes)

Projets de ISBSG avec les points extrêmes				
Langage	Nombre	Taille en points de fonction	Équation	R^2
Autres 3GL	3	290-330	Y= 24.967x - 5728.9	0.9972
Autres 4GL	31	110-6000	Y= 12.11x - 3272	0.6183
Access	17	200-1500	Y= -0.1049x + 840.84	0.0146
Autres ApG	16	2000-10000	Y= 5.7398x + 4502.6	0.3503
C	15	40-2500	Y= 4.0578x + 4288	0.1903
C++	21	70-1500	Y= 13.433x + 1346.4	0.6252
Clipper	4	200-2000	Y= 17.447x - 2750.3	0.9803
Cobol	106	0-5000	Y= 4.9496x + 5269.3	0.3666
Cobol II	21	80-2000	Y= 27.803x - 3593	0.9674

CSP	7	190-1500	Y= 7.7844x + 1912.8	0.6974
Easytrieve	8	70-250	Y= 3.1211x + 1246.9	0.195
Ideal	6	840-6000	Y= 2.201x + 7640.5	0.3844
Java	4	150-1000	Y= 9.2076x + 526.72	0.9235
Natural	41	20-3500	Y= 10.053x – 648.91	0.8577
Oracle	26	110-4300	Y= 6.2089x + 509.73	0.4279
PL/1	29	80-2600	Y= 11.069x + 46.774	0.2343
Powerbuilder	18	60-900	Y= 12.995x – 380.26	0.6615
SQL	20	280-4400	Y= 7.5781x – 271.24	0.6042
SQL Forms	3	150-1000	Y= 1.8032x + 992.48	0.5755
SQL Windows	4	150-600	Y= 6.7438x + 5347.6	0.663
Telon	23	70-1100	Y= 7.4233x + 650.92	0.8555
Visual basic	34	30-2300	Y= 9.6984x + 661.57	0.5612

Ce premier tableau (4.4) présente pour chaque langage le nombre de projets de l'échantillon, l'intervalle de la taille des projets en points de fonction, l'équation de régression et le coefficient de corrélation. Vu que la plupart des langages contiennent des projets dont le comportement est identique à celui de l'exemple de *Natural* précité, tous les projets qui nécessitent beaucoup d'effort et qui ont une grande taille font l'ob

afin d'éviter des biais et dans certains cas, les points extrêmes sont éliminés, comme présenté dans le tableau 4.5.

Tableau 4.5 : Liste des échantillons (sans leurs points extrêmes)

Sans les points extrêmes				
Langage	Nombre	Taille en points de fonction	Équation	R^2
Autres 3GL	3	290-330	Y= 24.967x – 5728.9	0.9972
Autres 4GL	8	110-950	Y= -3.2013x + 4699.5	0.3543
	12	951-5000	Y= 3.3387x + 6582.7	0.0861
Access	11	200-800	Y= 0.3091x + 623.56	0.1946
Autres ApG	7	200-1000	Y= 2.5521x + 933.61	0.3535
C	9	200-800	Y= 2.3421x + 2951.7	0.2908
C++	12	70-500	Y= 11.531x + 1197.1	0.1173
	5	750-1250	Y= -6.579x + 23003	0.0601
Clipper	3	234-500	Y= 12.874x - 1272.4	0.569
Cobol	60	60-400	Y= 10.837x + 299.13	0.4487
	32	401-3500	Y= 12.328x – 14.103	0.6462
Cobol II	9	80-180	Y= 16.396x – 92.346	0.457
	6	180-500	Y= 26.739x – 3340.8	0.6177
CSP	5	230-350	Y= 57.716x - 12172	0.9119

Easytrieve	6	70-200	Y= 17.028x – 630.17	0.8662
Ideal	3	840-3650	Y= 2.8787x + 2118.3	0.8014
Java	3	150-350	Y= 9.7534x + 408.58	0.3541
Natural	30 9	20-620 620-3500	Y= 6.1355x + 264.98 Y= 10.539x – 1404.9	0.475 0.7424
Oracle	19	100-2000	Y= 7.7803x – 1280.7	0.3918
PL/1	19 5	80-450 451-2550	Y= 8.3264x – 197.67 Y= 5.4914x – 65.349	0.6441 0.8699
Powerbuilder	12	60-400	Y= 1.9942x + 1560.1	0.1304
SQL	11 8	280-800 801-4500	Y= -0.4274x + 3831.2 Y= 9.2394x – 6064.3	0.0005 0.678
SQL Forms	3	150-1000	Y= 1.8032x + 992.48	0.5755
SQL Windows	4	150-600	Y= 6.7438x + 5347.6	0.663
Telon	18	70-650	Y= 5.5006x + 1046.1	0.7524
Visual basic	24	30-600	Y= 7.2487x + 52.477	0.4657

Dans le chapitre précédent, il a été question d'identifier des échantillons appropriés et dans celui-ci de développer des modèles d'échantillon homogènes pour mener à bien cette étude. Ces items seront maintenant utilisés afin de décrire et d'analyser les résultats obtenus. Cette analyse se fera en trois différentes phases. D'abord les résultats pour le modèle initial avec les points extrêmes et ceux pour les modèles développés sans les points extrêmes seront

es chapitres 5 et 6. Cela sera suivi dans le chapitre 7 d'une analyse basée sur l'erreur et sur le modèle de régression linéaire.

CHAPITRE V

ANALYSE DES RÉSULATS POUR LE MODÈLE INITIAL AVEC LES POINTS EXTRÊMES

> Dans cette première phase de l'analyse, tous les projets sont considérés dans la préparation de modèles d'estimation par langage de programmation, même ceux qui pourraient être considérés comme des points extrêmes soit par leur très grande taille par rapport à la majorité des projets, soit par leu
> développement (en termes d'effort) par rapport également à la majorité des autres projets.

Pour faciliter la lecture de ce texte, les exemples utilisés porteront sur les projets provenant
ation, soit *Natural*. Les analyses subséquentes portent cependant sur la majorité des langages de programmation disponibles dans la base de données qui nous était disponible, soit celle de l'International Software Benchmarking Standards Group - ISBSG. Ainsi, cette analyse débutera, par une comparaison de l'effort réel et de celui estimé par le progiciel SLIM, suivie d'une comparaison entre l'effort réel et celui estimé par l'environnement automatisé de ISBSG. Finalement, les résultats des deux processus d'estimation (SLIM et ISBSG) seront comparés. Il faut cependant noter que pour des raisons de disponibilité de données, SLIM a été utilisé dans sa globalité, c'est à dire qu'aucune portion ou sous-ensemble de SLIM n'a été sélectionnée pour les fins de comparaison. En effet, depuis plus de vingt ans, le fournisseur n'a jamais rendu les données de SLIM accessibles; il est ainsi impossible de faire des vérifications indépendantes.

D'autre part, afin de mieux évaluer la qualité des estimations de SLIM avec la base de données ISBSG, les projets de cette base sont estimés à même l'environnement automatisé de ISBSG. La tendance pourrait mener à croire qu'il existe un biais. En fait, ce n'est pas un biais, car, d'après Abran, expert dans le domaine, c'est la bonne façon
productivité. Il s'agit d'abord de développer un modèle d'estimation avec des données de projets antérieurs, et ensuite de l'évaluer par rapport à ces données réelles qui ont servi à le

des données autres que celles qui ont servi à le construire. Ce qui n'est pas possible dans le cas de SLIM parce que :

- Les données de SLIM ne sont pas connues (ni le nombre, ni la date, ni le domaine des projets n'est accessible);
- Le modèle lui-même n'est pas publié, c'est-à-dire que les équations paramétriques du logiciel restent inconnues, ce qui fait du progiciel SLIM une « boîte noire ».

Donc, la seule façon de l'évaluer c'est avec des données connues comme celles de ISBSG. Et en pratique, pour un produit commercial, c'est bien meilleur de travailler avec du connu et du transparent, comme le suggère l'approche scientifique.

5.1 COMPARAISON DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR SLIM

Tout d'abord, afin que la comparaison soit menée à bien, la différence entre l'effort réel de chaque projet et celui estimé par SLIM est calculée. Lorsque cette différence a une valeur négative, ceci exprime une surestimation de l'effort du projet, et vice versa. Certains langages ont la majorité de leurs projets surestimés ou sous-estimés, alors que d'autres ont un pourcentage égal (50%) entre la surestimation et la sous-estimation. En fait, vu le type d'analyse utilisée, c'est-à-dire la comparaison des efforts, les 50% sont le chiffre idéal auquel il faudrait s'attendre.

Ainsi, pour le cas du langage de quatrième génération *Natural* avec 41 projets dans la base de données ISBSG, 59% des projets ont un effort sous-estimé par le progiciel SLIM et, évidemment, les 41% restants sont surestimés. Cependant, qu'ils soient positifs ou négatifs, les écarts sont plus ou moins grands dépendemment des échantillons par type de langages de programmation. Par exemple, dans l'échantillon de projets en langage *Natural*, pour une taille de 215 points de fonction, un projet bien spécifique a un effort réel de 560 PHR et un effort sous-estimé de 551 PHR, donc une différence d'estimation de 8.9 heures, soit une erreur relative de 1.59%. De même, le projet dans cet échantillon qui a une taille de 2 171 points de fonction, a un effort réel de 19 699 PHR et un effort surestimé de 117 824.6 PHR,

donc une différence de 98 125.6 heures, soit de 498.12%. Ce qui explique que la droite de régression de l'effort estimé des projets suit une tendance différente de cel
mais, toujours est-il que la corrélation reste positive entre l'effort du projet et sa taille comme le montre la figure 5.6. L'équation de la régression précitée est établie comme suit :

$$y = (a * x) + b$$

Ce qui donne par analogie :

$$Effort = (a * Taille) + b$$

Ainsi, l'analyse de la régression de l'effort estimé par SLIM avec un « n » égal à 41 projets donne un coefficient de corrélation (R^2) équivalent à 0.2264 (figure 5.6), et l'équation suivante:

$$Nombre\ d'heures/personnes = (17.776 * Nombre\ de\ points\ de\ fonction) + 2\ 068.5$$

Ce modèle de SLIM présente un écart-type considérable de 24 839.95 heures et les estimations ponctuelles, bien qu'utiles, ne fournissent aucune information concernant la -à-dire qu'elles ne tiennent pas compte de l'erreur possible dans l'estimation. De ce fait, un intervalle est associé aux estimations de SLIM pour leur permettre d'englober, avec une certaine fiabilité, la vraie valeur de l'effort, c'est-à-dire l'effort réel. Il s'agit de l'intervalle de confiance. Ainsi, avec un niveau de confiance de 99%, les efforts réels des projets de *Natural* varient de ± 9 992.56 heures des estimations faites par SLIM. Également, pour 27% des projets de ce langage *Natural*, leur effort réel est au-delà des limites de leurs intervalles de confiance respectifs.

En effet, pour une même taille, l'effort estimé par SLIM a un comportement considérablement différent de l'effort réel sur la droite de régression, même avec une

corrélation légèrement positive entre l'effort réel et la taille des projets. Ce qui explique l'importance de l'erreur relative moyenne de 371.11% pour l'ensemble de l'échantillon; ce dernier pourcentage indique qu'il existe des écarts considérables entre les estimés et les

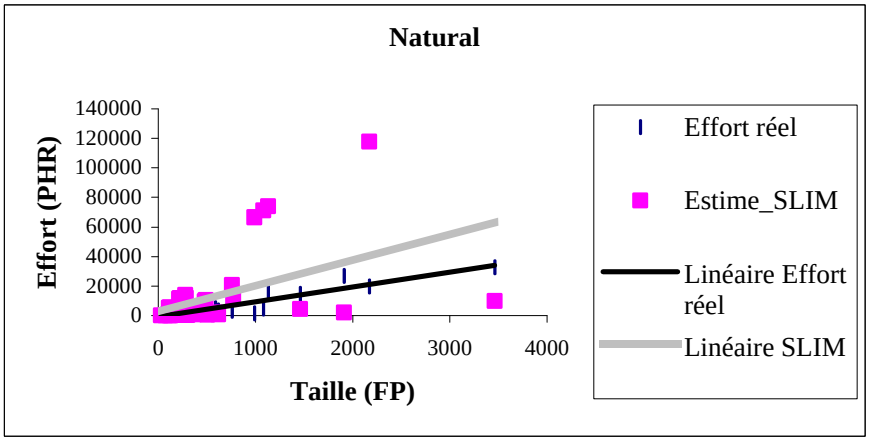


Figure 5.6 : Effort réel et estimé par SLIM en fonction de leur taille

Cette figure 5.6 représente l'échantillon des 41 projets développés en *Natural*. Cependant, « *Natural* » est un langage parmi tant d'autres utilisés dans cette étude. Ces langages sont de trois types différents, à savoir, des langages procéduraux de la 3^{e} génération, des outils de la 4^{e} génération et de ceux des générateurs d'application (Application Generators APG). Chaque logiciel appartient à un type de langage bien précis, à l'exception de *Java*, pour qui 50% des projets sont développés en 3GL et les 50% restants sont développés dans un environnement de 4GL.

5.2 COMPARAISON PAR GÉNÉRATION DE LANGAGE DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR SLIM

Les tableaux qui suivent font la récapitulation de tous les projets des différents types de langages et donnent un bref aperçu sur les comparaisons entre l'effort réel et l'estimation

faite par le progiciel SLIM, et entre ce même effort réel et celui estimé par l'environnement

r une meilleure compréhension, il faut noter que les types de *autres 3 GL »*, « *autres 4 GL »* et « *autres APG »* présentés dans tous les tableaux subséquents ne représentent pas un langage en particulier, mais, une catégorie de langages de la troisième et de la quatrième génération en plus de ceux des générateurs d'application, dont les noms ne sont pas identifiés par les développeurs des projets.

Tableau 5.6 : Sous-estimation et surestimation - langages 3 GL

Types de langage 3 GL	Portion sous-estimée		Portion surestimée		Portion majoritaire	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)	SLIM	ISBSG
Autres 3GL	67	67	33	33	Sous-estimé	Sous-estimé
C	27	33	73	67	Surestimé	Surestimé
C++	90	33	10	67	Sous-estimé	Surestimé
Cobol	44	26	56	74	Surestimé	Surestimé
Cobol II	86	76	14	24	Sous-estimé	Sous-estimé
Java	100	0	0	100	Sous-estimé	Surestimé
PL1	55	21	45	79	Sous-estimé	Surestimé

Ce tableau montre que la plupart des estimations faites par SLIM (avec trois paramètres seulement, soient la taille fonctionnelle, la durée, et le langage de programmation) donnent des sous-estimations par rapport à l'effort réel. En effet, 71% des langages de la troisième génération ont des projets dont les efforts sont sous-estimés. Et seuls 2 langages (*C* et *Cobol*) sur 7 contiennent des projets dont les efforts sont surestimés et l'un l'est à l'excès, c'est-à-dire que l'estimation faite par SLIM est de loin supérieure à l'effort réel. Il s'agit du langage *C* et cela justifie le fait que la tendance de la droite de régression de l'eff
étant croissante, soit carrément différente de celle de l'effort réel.

Par ailleurs, pour la majorité (93%) des langages de la quatrième génération (tableau 5.7), l'effort estimé par SLIM est plus petit que l'effort réel, cependant, comme
logiciel *Access*, l'écart est moindre que celui existant dans les projets développés dans tous les autres langages utilisés pour cette recherche. Ainsi, à part, un projet extrême qui a été trop

surestimé, l'ensemble de l'effort estimé suit la même tendance que l'effort réel, traduit par la droite de régression linéaire.

Tableau 5.7 : Sous-estimation et surestimation - langages 4 GL

Types de langage 4 GL	Portion sous-estimée		Portion surestimée		Portion majoritaire	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)	SLIM	ISBSG
Autres 4GL	77	42	23	58	Sous-estimé	Surestimé
Access	88	47	12	53	Sous-estimé	Surestimé
Java	100	50	0	50	Sous-estimé	Égal
Clipper	75	50	25	50	Sous-estimé	Égal
Easytrieve	100	50	0	50	Sous-estimé	Égal
CSP	86	43	14	57	Sous-estimé	Surestimé
Ideal	83	50	17	50	Sous-estimé	Égal
Natural	59	54	41	46	Sous-estimé	Sous-estimé
Oracle	35	35	65	65	Surestimé	Surestimé
Powerbuilder	100	50	0	50	Sous-estimé	Égal
SQL	90	40	10	60	Sous-estimé	Surestimé
SQL Forms	100	67	0	33	Sous-estimé	Sous-estimé
SQL Windows	100	50	0	50	Sous-estimé	Égal
Visual Basic	74	26	26	74	Sous-estimé	Surestimé

Tableau 5.8 : Nature de l'estimation des langages en APG

Types de langage APG	Portion sous-estimée		Portion surestimée		Portion majoritaire	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)	SLIM	ISBSG
Autres APG	81	19	19	81	Sous-estimé	Surestimé
Telon	100	39	0	61	Sous-estimé	Surestimé

D'autre part, le tableau 5.8 ci-dessus montre que la totalité des langages développés en APG ont des projets dont les estimations de SLIM sont majoritairement sous-évaluées. Il va sans dire qu'à l'aide de ces deux langages uniquement, une généralisation ne peut être faite. *autres APG* », à elle seule, a une portion sous-estimée de SLIM de 81%.

5.3 COMPARAISON DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR ISBSG

Stroian (1999) a développé pour le Laboratoire de recherche en gestion des logiciels de l'UQAM un environnement automatisé pour un processus transparent d'estimation fondé sur la base de données historiques de l'International Software Benchmarking Standards Group (ISBSG). Il représente une contribution intéressante à ISBSG en donnant une valeur ajoutée à

Les estimations faites par cet environnement automatisé de ISBSG, tout comme celles faites par SLIM, sont différentes de l'effort réel. Pour ce qui est de *Natural*, 54% des projets ont des efforts sous-estimés par l'environnement automatisé de ISBSG et de ceux-ci, 37% ont été également sous-estimés par SLIM, cependant, il est à noter que les écarts ne sont pas les mêmes. Ainsi, un effort sous-estimé par ISBSG, ne l'est pas nécessairement par SLIM. De ce fait, le comportement d'un projet bien déterminé peut être différent d'un modèle d'estimation à l'autre. De façon plus explicite, les estimations faites
automatisé de ISBSG sont indépendantes, et l'une n'influence pas l'autre.

De plus, avec un facteur d'équivalence égal à 40 (Appendice B), *Natural* présente 43% de ses projets qui sont surestimés par ISBSG et de ceux-ci, 58% le sont par SLIM. Et pour un facteur d'équivalence égal à 70, un peu moins de la moitié, c'est-à-dire 46% des projets de *Natural* sont surestimés par ISBSG et tous ces projets le sont aussi par SLIM. Cela confirme le fait que les estimations de SLIM et de ISBSG soient indépendantes, l'une de l'autre, mais aussi, indique que le facteur d'équivalence a une influence sur les estimations effectués par SLIM (Appendice C).

Par ailleurs, le tableau 5.7 montre que sur les 14 types de langages, deux ont des projets dont les efforts sont majoritairement sous-estimés. Il s'agit de *Natural* et de *SQL Forms*. Parallèlement, six autres outils de la quatrième génération ont des projets dont le pourcentage

-estimation de l'effort. Par exemple, pour *Clipper*,

50% des projets ont des efforts sous-estimés, et les 50% restants sont surestimés. Il en est de même pour les langages *Ideal, Powerbuilder, Java, Easytrieve et SQL Windows*.

Sinon, en ce qui concerne les langages de la troisième génération (tableau 5.6), 71%, c'est-à-dire cinq langages sur sept contiennent des projets dont l'ensemble des efforts estimés par ISBSG est supérieur à l'effort réel. Cependant, la catégorie « *autres 3GL* » présente 2/3 de ses projets qui sont légèrement sous-estimés, c'est-à-dire qu'ils ont un effort réel sensiblement pareil à celui estimé. En plus de ce dernier, *Cobol II* se démarque également du lot par le fait que les efforts de ses projets estimés par ISBSG soient généralement sous-évalués.

Pour ce qui est de l'APG et de ses outils, tous ces langages sont caractérisés par des efforts surestimés. À l'évidence même, avec juste ces deux outils, aucune conclusion globale ne peut -il que la catégorie « *autres APG* » elle-même présente 81% de ses projets avec un effort surestimé. Serait-ce une indication pour l'ensemble de ses outils par rapport à l'estimation faite par ISBSG? Ce n'est sûrement pas le cas, car, comme il a été cité précédemment dans les outils de la troisième générati *autres 3GL* » a des projets dont les efforts sont sous-estimés par ISBSG; et paradoxalement, la majorité des logiciels de la troisième génération présentent des projets avec des efforts dont les comportements sont différents de celui de l' *autres 3 GL* »; c'est-à-dire qu'ils sont surestimés. Ainsi, l'on ne peut généraliser la nature des estimations faites par ISBSG pour une catégorie de langages, par le comportement de ce même langage, mais, la supposition els de cette génération ont des attitudes distinctes de celles du langage lui-même peut être valable.

Somme toute, de façon plus générale, les estimations de ISBSG sont différentes de l'effort réel. Il en est de même pour celles de SLIM, compte tenue de la base de comparaison. Cependant, en ce qui concerne ce dernier, la tendance majeure est à la sous-estimation pour l'ensemble des logiciels; et les efforts estimés par ISBSG présentent l'effet contraire à savoir

D'après ce qui précède, les efforts des projets des différents langages ont été estimés à l'aide de deux modèles de productivité soient le progiciel SLIM et l'environnement automatisé de ISBSG. Les résultats d'estimation de chacun de ces deux modèles ont été comparé
réel respectif de chaque projet. Il est maintenant intéressant de comparer les efforts estimés des deux modèles. Ce qui fait l'objet du paragraphe subséquent.

5.4 COMPARAISON DES RÉSULTATS DES DEUX MODÈLES D'ESTIMATION : SLIM ET ISBSG

En considérant toujours l'exemple de *Natural*, le tableau 5.9 montre que 27% des estimations faites par ISBSG sont plus petites que celles de SLIM, donc nécessairement, les 73% autres sont supérieures à celles de SLIM. Ce qui justifie, que pour le cas de *Natural* la majorité de ses projets ont une estimation de ISBSG supérieure à celle de SLIM. Et bien que les pourcentages diffèrent d'un langage à l'autre, la quasi-totalité des langages ont le même comportement que *Natural*. Il s'agit en effet des deux langages développés en APG et du 6/7 des outils en 3GL. Pour ce qui est des outils en 4GL, seul *Oracle* présente le contraire.

Tableau 5.9 : Comparaison des estimés de ISBSG et de SLIM par type de langages

Types de langage 3 GL	Portion de ISBSG < SLIM (%)	Conclusion
Autres 3GL	33	ISBSG > SLIM
C	67	ISBSG < SLIM
C++	5	ISBSG > SLIM
Cobol	41	ISBSG > SLIM
Cobol II	48	ISBSG > SLIM
Java	0	ISBSG > SLIM
PL1	28	ISBSG > SLIM

Types de langage 4GL	Portion de ISBSG < SLIM (%)	Conclusion
Autres 4GL	29	ISBSG > SLIM
Access	12	ISBSG > SLIM
Clipper	0	ISBSG > SLIM
Java	0	ISBSG > SLIM
Easytrieve	0	ISBSG > SLIM
CSP	0	ISBSG > SLIM
Ideal	17	ISBSG > SLIM
Natural	27	ISBSG > SLIM

Oracle	62	ISBSG < SLIM
Powerbuilder	0	ISBSG > SLIM
SQL	10	ISBSG > SLIM
SQL Forms	0	ISBSG > SLIM
SQL Windows	0	ISBSG > SLIM
Visual Basic	21	ISBSG > SLIM

Types de langage APG	Portion de ISBSG < SLIM (%)	Conclusion
Autres APG	0	ISBSG > SLIM
Telon	0	ISBSG > SLIM

Pour une comparaison plus précise, il faut maintenant analyser l'erreur relative moyenne de chaque type de langage (tableau 5.10).

Tableau 5.10 : Comparaison des estimés de SLIM et de ISBSG par l'erreur relative moyenne

Types de langage 3 GL	Erreur relative moyenne (MMRE)		Conclusion
	SLIM (%)	ISBSG (%)	
Autres 3GL	39.82	2.53	ISBSG < SLIM
C	826.68	144.48	ISBSG < SLIM
C++	71.09	76.38	ISBSG > SLIM
Cobol	237.36	268.83	ISBSG > SLIM
Cobol II	19.21	146.55	ISBSG > SLIM
Java	87.09	62.59	ISBSG < SLIM
PL1	227.38	85.16	ISBSG < SLIM

Types de langage 4GL	Erreur relative moyenne (MMRE)		Conclusion
	SLIM (%)	ISBSG (%)	
Autres 4GL	126.34	266.85	ISBSG > SLIM
Access	118.79	22,25	ISBSG < SLIM
Clipper	89.82	87,63	ISBSG < SLIM
Java	92.58	18	ISBSG < SLIM
Easytrieve	87.78	44.45	ISBSG < SLIM
CSP	66.28	62,92	ISBSG < SLIM
Ideal	107.61	59,65	ISBSG < SLIM
Natural	371.11	74.39	ISBSG < SLIM
Oracle	505.14	142.65	ISBSG < SLIM
Powerbuilder	89.61	63.85	ISBSG < SLIM
SQL	127.59	179.85	ISBSG > SLIM
SQL Forms	61.48	44.58	ISBSG < SLIM
SQL Windows	95.04	9.19	ISBSG < SLIM
Visual Basic	116	117.30	ISBSG > SLIM

autres

APG ». Ce qui illustre que dans l'ensemble des langages, la plupart ont des projets dont les efforts sont mieux estimés par ISBSG que par SLIM; non seulement parce que les erreurs calculées pour ISBSG sont inférieures à celle de SLIM, mais aussi, le fait que les erreurs pour SLIM soient généralement beaucoup plus élevées que celles de ISBSG. Ce qui permet de juger de la qualité moindre des estimations de SLIM, par rapport à celles de ISBSG.

Cependant, bien que l'environnement automatisé de ISBSG paraît plus fia -à-dire que ses estimations se rapprochent plus de la réalité, les erreurs restent quand même hautes. Ainsi, sauf pour les langages tels que la catégorie « *autres 3GL* », *Access*, *Java* de la quatrième génération et *SQL Windows*, ce modèle d'estimation n'est pas acceptable. En effet, seuls ces quatre langages ont des erreurs relatives moyennes de ISBSG inférieures à 25%. Et, dans le cas de SLIM, seules les évaluations faites pour Cobol II rencontrent ce critère.

De façon plus globale, avec ce modèle où tous les projets sont considérés, l'environnement automatisé de ISBSG se révèle meilleur estimateur que l'outil SLIM. Cependant, avant de généraliser cette conclusion, une pareille étude sera faite pour les modèles développés sans qui introduit le prochain chapitre.

CHAPITRE VI

ANALYSE DES RÉSULATS POUR LES MODÈLES DÉVELOPPÉS SANS LES POINTS EXTRÊMES

Cette phase, fait référence aux modèles d'échantillons développés dans le chapitre 4 après l'élimination des points extrêmes. Ainsi,
partie de l'analyse de données, ces modèles développés avec des échantillons plus homogènes sont utilisés pour les fins de cette analyse.

6.1 COMPARAISON DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR SLIM

Les tableaux subséquents montreront que pour certains langages il est possible de construire deux modèles distincts, après avoir identifié par analyse visuelle graphique qu'il y avait deux ensembles distincts de projets ayant des comportements plus homogènes dans la relation Taille Fonctionnelle/Effort. C'est le cas de *Natural* qui est toujours utilisé comme exemple. Initialement, cet échantillon contenait 41 projets, mais, deux d'entre eux ont été enlevés, l'un 138 heures/personne) et l'autre à cause de sa petite taille (168 points de fonction) par rapport à l'effort (4 622 heures/personne) fourni lors du développement. Et donc, il reste maintenant 39 projets disponibles, pour lesquels 2 sous-ensembles ont été identifiés par analyse graphique visuelle .

Son premier modèle est construit en utilisant un sous-échantillon de 30 projets dont la taille de chacun varie entre 20 et 620 points de fonction inclusivement, et son deuxième est construit en utilisant l'autre sous-échantillon de 9 gros projets qui ont une taille comprise entre 621 et 3 500 points de fonction.

Comme pour le modèle initial avec les points extrêmes, ce n'est pas l'erreur relative moyenne, mais l'écart absolu entre l'effort réel et celui estimé par SLIM qui est calculé. Ce *Natural* une sous-estimation de 60% pour son premier modèle, et une surestimation de 56 % pour son ensemble de 9 projets. Comme dans le cas du modèle avec

tives pour les deux ensembles de projets :

- pour les projets dont la taille est comprise entre 20 et 620 points de fonction l'équation est la suivante :

$$Nombre\ d'heures/personnes = (3.3 * Nombre\ de\ points\ de\ fonction) + 2\ 604$$

Avec cette équation de régression, ce premier ensemble de 30 projets présente un coefficient de corrélation (R^2) égal à 0.0134 (donc très faible), et la relation reste toujours positive entre la taille du projet et l'effort estimé par SLIM (figure 5.7). Par ailleurs, son écart-type est de 4 404.88 heures ce qui donne une variation des efforts réels des différents projets de ± 2 071.83 heures des estimations de SLIM. De plus, 47 % des projets seulement de *Natural* ont leur effort réel situé hors de cet intervalle de confiance.

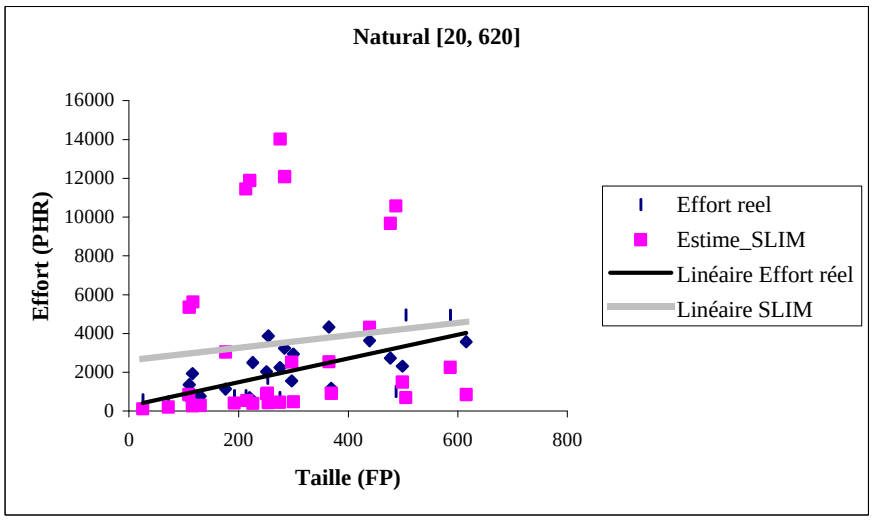


Figure 6.7: L'effort des 30 projets de *Natural* en fonction de leur taille (plus petite que 620 PF)

ces projets varient de ± 35 381.73 heures des efforts estimés par SLIM. Et seuls 5 sur 9 projets ont un effort réel encadré par les limites de leurs intervalles de confiance respectifs.

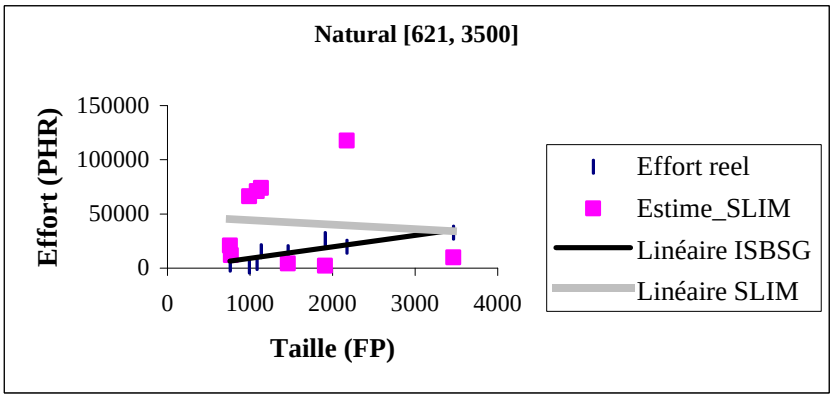


Figure 6.8 : L'effort des 9 gros projets de *Natural* en fonction de leur taille (plus grande que 620 PF)

D'après les deux figures précédentes les estimations avec SLIM sur les variables de taille, de durée et de langage de programmation sont ainsi très loin d'être égales à l'effort réel, ce qui explique les valeurs respectives de l'erreur moyenne relative de ces deux modèles. Elle est de 248.01% pour le premier modèle, et de 850.63 % pour le deuxième. Donc, les écarts moyens des estimations des efforts de cet ensemble de 9 projets sont 3.4 fois plus grands que l'erreur relative moyenne des estimations des efforts des 30 projets. D'autre part, étant donné l'intervalle de la taille de l'ensemble des projets, aucune interprétation ne peut être faite pour les projets de *Natural* ayant une taille inférieure à 20 points de fonction ou supérieure à 3 500

points de fonction, compte tenu de la non disponibilité des données dans la base de données ISBSG pour ces intervalles.

De façon plus générale, les deux modèles du langage *Natural* ont des estimations de nature distincte, à savoir une sous-estimation des efforts des projets du premier ensemble de 30 projets et une surestimation de ceux dont la taille est comprise entre 621 et 3 500 points de fonction.

Tableau 6.11 : Nature de l'estimation des langages en 4 GL

Types de langage 4 GL [Intervalle de taille en PF]	Nombre de projets	Portion sous-estimée		Portion surestimée		Portion majoritaire	
		SLIM (%)	ISBSG (%)	SLIM (%)	ISBSG (%)	SLIM	ISBSG
4GL [110, 950]	8	75	33	25	67	Sous-estimée	Surestimée
[951, 5000]	12	75	38	25	62	Sous-estimée	Surestimée
Access [200,800]	11	81	45	9	55	Sous-estimée	Surestimée
Clipper [234, 500]	3	67	67	33	33	Sous-estimée	Sous-estimée
Java 188	1	100	100	0	0	Sous-estimée	Surestimée
Easytrieve [70, 200]	6	100	50	0	50	Sous-estimée	Égale
CSP [230, 350]	5	80	80	20	20	Sous-estimée	Sous-estimée
Ideal [840, 3650]	3	67	33	33	67	Sous-estimée	Surestimée
Natural [20, 620]	30	60	43	40	57	Sous-estimée	Surestimée
[621, 3500]	9	44	44	56	56	Surestimée	Surestimée
Oracle [100, 2000]	19	26	37	74	63	Surestimée	Surestimée
Powerbuilder [60, 480]	12	100	58	0	42	Sous-estimée	Sous-estimée
SQL [280, 800]	11	81	55	9	45	Sous-estimée	Sous-estimée
[1350, 4500]	8	87	25	13	75	Sous-estimée	Surestimée
SQL Forms [150, 1000]	3	100	67	0	33	Sous-estimée	Sous-estimée
SQL Windows [150, 600]	4	100	50	0	50	Sous-estimée	Égale
Visual Basic [30, 600]	24	75	42	25	58	Sous-estimée	Surestimée

D'ailleurs, d'après le tableau 6.11 ci-dessus des langages de 4GL, il est le seul à avoir deux e comportement des estimations diffère l'un de l'autre. Effectivement, en plus

de *Natural*, les projets de « *autres 4GL* » catégorisés par ISBSG et ceux de *SQL* possèdent également deux modèles. Mais, à la différence des projets de *Natural*, les deux ensembles de projets distincts de ces deux modèles sont sous-estimés, et ils sont une partie intégrante des 15 modèles sur 17 dont les estimations de SLIM sont sous-évaluées par rapport à l'effort réel.

Cependant, un langage sur ces 15 est particulier. Il s'agit de *Java* qui ne contient qu'un seul projet développé en 4GL, puisque l'autre représentait un point extrême. Donc, une conclusion générale ne peut être faite en ce qui concerne ce dernier langage, mais, il fait partie de l'ensemble des outils de la quatrième génération qui, mis à part le langage *Oracle*, ont des projets dont les efforts sont majoritairement sous-estimés par SLIM. Ce qui rejoint la conclusion de la partie précédente où tous les projets sont considérés sans exception (modèle mes).

Ce qui nous mène alors à conclure que, sauf pour certains cas particuliers précités, les estimations de l'effort faites par le progiciel SLIM pour les langages de la quatrième génération sont en général inférieures à l'effort réel, indépendamment de la taille des projets.

Pour ce qui est des projets développés dans un langage de troisième génération, le tableau ci-dessous montre que 64% de ces derniers ont des efforts sous-évalués par SLIM. De plus, il semblerait que la nature de l'estimation soit indépendante de la catégorie de taille de chaque outil de 3GL, à l'exception du langage *PL1*. Ce dernier a des efforts sous-estimés pour les projets dont la taille est comprise entre 80 et 450 points de fonction, et présente le contraire pour les projets ayant une taille supérieure ou égale à 550, et inférieure ou égale à 2 550 points de fonction. Cette surestimation n'est pas seulement à l'égard d'une partie des projets *PL1*; elle caractérise également les efforts des projets de deux autres langages tels que *C* et *Cobol* dans leur totalité, peu importe la catégorie de taille à laquelle appartiennent les projets.

Tableau 6.12 : Nature de l'estimation des langages en 3GL

-estimés, même si les pourcentages de sous-évaluation diffèrent d'un modèle à un autre. Également, le fait de développer des modèles pour certains langages tels *autres 4GL* », *Natural*, *SQL*, *C++*, *Cobol*, *Cobol II* et *PL1*, ne fait pas de la sous-estimation antérieure une surestimation.

importantes et SLIM ne peut pas être considéré comme un bon modèle d'estimation lorsque seulement les variables taille, durée et langage de programmation sont utilisées. Ces résultats ont des similarités avec les résultats de l'étude

Kemerer en 1987, le pourcentage moyen de l'erreur des estimations de SLIM était de 772%. Afin de déterminer la qualité des estimations de différents modèles algorithmiques (SLIM, es informations de 15 projets, et il a trouvé qu'avec SLIM les erreurs sont biaisées et que l'effort est surestimé dans la totalité des 15 cas.

6.2 COMPARAISON DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR ISBSG

Comme dans le cas des estimations de SLIM, la nature des écarts de chacun des deux modèles du langage *Natural* est la même, c'est-à-dire que cette fois-ci, c'est l'ensemble des 30 projets qui est surestimé à 57% par ISBSG et celui des neuf projets est également surévalué à 56%. Hormis ces deux modèles de *Natural*, cinq autres langages ont des projets dont les efforts estimés sont inférieurs à l'effort réel. Il s'agit de *Clipper*, *CSP*, *Powerbuilder*, *SQL Forms* et de l'ensemble des projets de *SQL* dont la taille est comprise entre 280 et 800 points de fonction. Sinon, à l'exception de Easytrieve et de SQL Windows pour qui le pourcentage de la sous-estimation est identique à celui de la surestimation, c'est-à-dire 50% chacun, tous les 59% restants des langages montrent une surévaluation de leur effort. Ce qui signifie que la majorité des estimations faites par ISBSG pour les langages développés en 4GL est plus élevée que l'effort réel.

Cette surestimation concerne également un peu plus de la moitié des projets (55%) des langages de la troisième génération. Effectivement, 4 ensembles de projets sur 11 ont des efforts sous-évalués par ISBSG et un autre a 50% de surestimation et 50% de sous-évaluation. Ce dernier s'agit des projets de *Cobol II* ayant une taille entre 181 et 500 points de fonction. Qui plus est, ils ont tous les quatre des efforts estimés par ISBSG plus petits que leurs efforts réels respectifs. Cependant, cela peut ne pas être significatif, vu que seuls 5

ensembles de projets sur 13 (langages de troisième génération et outils de l'APG) ont un comportement identique – peu importe que ce soit une surestimation ou une sous-estimation – par rapport aux estimations des deux sortes de modèles, à savoir, SLIM et ISBSG. Il s'agit de *« autres 3 GL »* [290, 330][7], de *Cobol II* [80,180], de tous les projets de *Cobol*, et de l'ensemble « *autres APG* » [200, 1000], les chiffres entre crochets étant l'étendue de la taille en points de fonction.

6.3 COMPARAISON DES RÉSULTATS DES DEUX MODÈLES D'ESTIMATION : SLIM ET ISBSG

Le deuxième modèle de *Natural* qui contient des projets dont la taille varie entre 621 et 3500 points de fonction est caractérisé par des estimations de l'environnement automatisé de ISBSG inférieures à celles de SLIM. C'est aussi le cas de 3/11 des outils de 3 GL et du langage de la quatrième génération *Oracle*.

Tableau 6.14 : Comparaison des estimés de ISBSG et de SLIM par type de langage

Types de langage 3 GL	Nombre de projets	Portion de ISBSG < SLIM (%)	Conclusion
Autres 3GL [290, 330]	3	33	ISBSG > SLIM
C [200, 800]	9	89	ISBSG < SLIM
C++ [70, 500]	12	17	ISBSG > SLIM
C++ [750,1250]	5	0	ISBSG > SLIM
Cobol [60, 400]	60	53	ISBSG < SLIM
Cobol [401, 3500]	32	48	ISBSG > SLIM
Cobol II [80, 180]	9	0	ISBSG > SLIM
Cobol II [181,500]	6	0	ISBSG > SLIM
Java [150, 350]	2	0	ISBSG > SLIM
PL1 [80, 450]	19	37	ISBSG > SLIM
PL1 [550, 2550]	5	60	ISBSG < SLIM

Types de langage 4GL	Nombre de projets	Portion de ISBSG < SLIM (%)	Conclusion
Autres 4GL [110, 950]	8	17	ISBSG > SLIM
Autres 4GL [951, 5000]	12	13	ISBSG > SLIM
Access [200, 800]	11	9	ISBSG > SLIM

[7] «*Autres 3 GL*» [290, 330] signifie les projets développés dans la catégorie de langages « *autres 3 GL*» et dont la taille de chacun de ces projets est comprise entre 290 et 330 points de fonction.

les erreurs relatives moyennes des estimations de ces deux logiciels d'estimation doivent être

Tableau 6.15 : Comparaison des estimés de SLIM et de ISBSG par l'erreur relative moyenne

Types de langage 3 GL	Nombre de projets	Erreur relative moyenne (MMRE)		Conclusion
		SLIM (%)	ISBSG (%)	
Autres 3GL [290, 330]	3	39.82	2.53	ISBSG<SLIM
C [200, 800]	9	1004.60	72.10	ISBSG<SLIM
C++ [70, 500]	12	64.89	88.21	ISBSG>SLIM
C++ [750,1250]	5	90.28	20.16	ISBSG<SLIM
Cobol [60, 400]	60	305.39	43.67	ISBSG<SLIM
Cobol [401, 3500]	32	604.58	168.33	ISBSG<SLIM
Cobol II [80, 180]	9	76.05	36.07	ISBSG<SLIM
Cobol II [181,500]	6	64.33	75.62	ISBSG>SLIM
Java [150, 350]	2	91.12	36.30	ISBSG<SLIM
PL1 [80, 450]	19	176.98	34.11	ISBSG<SLIM

s uniquement dans 7 cas sur 30, soit pour *C++* [70, 500], *Cobol II* [181, 500], les deux ensembles de la catégorie « *autres 4GL* », *Clipper* [234, 500], et pour tous les projets développés en *SQL* (*SQL* [280, 800] et *SQL* [1350, 4500]). Mis à part le langage *Clipper*, tous les autres susmentionnés ont eu le même comportement dans le modèle initial avec les points extrêmes. Cependant, chacun des deux ensembles de *Cobol II* et de *C++* a un comportement différent de l'autre. L'on peut donc déduire qu'en ce qui concerne ces deux langages, l'effort est mieux estimé par SLIM pour les projets développés en *C++* ayant une

SQL seront toujours mieux estimés par SLIM que par ISBSG, contrairement à *SQL Windows* et *SQL Forms*.

Par ailleurs, l'environnement automatisé de ISBSG est considérablement plus fiable que SLIM pour la majorité des langages. Et, il l'est encore plus pour certains dont l'erreur relative : il s'agit de *Access*, de *Telon.*, et à titre indicatif et non significatif (vue la taille petite des échantillons), il y a la catégorie « *autres 3GL* » pour les outils de la troisième génération, *Easytrieve* et *SQL Windows* pour ceux de la quatrième génération.

Hormis ces derniers et l'ensemble « *autres 4GL* » [110, 950], ni l'environnement automatisé de ISBSG, ni SLIM sont des modèles acceptables pour l'estimation des efforts des projets. explique par certains pourcentages trop élevés du tableau 6.15. En effet, la plupart des langages ont des erreurs relatives moyennes de plus de 25%. Ce qui juge de la précarité des deux moyens d'estimation pour ces derniers langages, car, pour qu'un modèle soit acceptable, il faudrait que les estimations faites par celui-ci soient inférieures ou égales à 0.25. Qui plus est, plus faible est l'erreur relative moyenne, meilleur est le modèle

CHAPITRE VII

ANALYSE DES RÉSULTATS BASÉE SUR LES MODÈLES DU COÛT UNITAIRE MOYEN ET SUR LA RÉGRESSION LINÉAIRE

Afin de s'assurer de la validité des résultats des deux chapitres précédents, d'autres tests sont effectués dans celui-ci, sur les mêmes données de ISBSG compte tenus des deux modèles, à savoir celui qui contient la totalité des projets et l'autre qui est composé de plusieurs ensembles de projets développés à l'aide

Il s'agit du test basé sur la régression linéaire. Mais tout d'abord, la racine carrée de l'erreur relative moyenne ($\overline{\text{RMS}}$ ou RRMS) et le niveau de prédiction à 25% (PRED(0.25)) sont calculés pour tous les projets.

7.1 ANALYSE DE L'ERREUR À PARTIR DU $\overline{\text{RMS}}$ ET DU (PRED(0.25)

Les tableaux qui suivent montrent les résultats en pourcentage du $\overline{\text{RMS}}$ et du PRED(0.25). Le premier (tableau 7.16) présente les pourcentages du modèle avec la globalité des projets sans distinction.

Tableau 7.16 : Pourcentage du niveau de prédiction et de la racine carrée de l'erreur relative moyenne de la totalité des projets

Types de langage	RRMS		PRED (0.25)	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)
Autres 3GL	65	2	67	100
Autres 4GL	199	110	3	13
Access	262	30	0	65
Autres APG	155	107	6	13
C	1054	75	13	7
C++	117	53	10	33
Clipper	122	15	0	50
Cobol	826	130	8	14

Cobol II	163	47	10	19
CSP	89	40	14	29
Easytrieve	99	40	0	25
Ideal	133	36	0	33
Java (4GL)	153	21	0	50
Java (3 GL)	53	18	0	50
Natural	490	54	2	32
Oracle	483	105	4	31
PL1	591	208	10	28
Powerbuilder	132	56	6	11
SQL	147	69	0	10
SQL Forms	81	30	0	33
SQL Windows	96	10	0	0
Telon	117	28	0	61
Visual Basic	173	89	0	24

Ainsi, pour l'ensemble des langages de ce dernier, la qualité meilleure des estimations de SLIM ne passe pas inaperçue. Réellement, peu importe la différence, la racine carrée de l'erreur relative moyenne des évaluations de SLIM est toujours plus élevée que celle de ISBSG. Et, d'un autre côté, son PRED(0.25) est toujours plus faible que celui de ISBSG, sauf pour *C* pour qui c'est le contraire, et pour *SQL Windows* qui a un niveau de prédiction de SLIM égal à celui de ISBSG.

Des résultats comparatifs similaires sont observés pour les modèles de langages développés sans les projets extrêmes. Cela se voit dans le tableau 7.17 où les calculs ont été effectués par intervalle de taille et où les types de langages sont considérés.

Tableau 7.17 : Pourcentages du RRMS et du PRED (0.25) des langages

Types de langage 4 GL [Taille]	RRMS		PRED (0.25)	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)
Autres 4GL [110, 950]	107	37	0	58
[951, 5000]	110	77	13	38
Access [200,800]	341	15	0	91
Clipper [234, 500]	101	43	0	33
Java 188	124	49	0	0
Easytrieve [70, 200]	103	19	0	83
CSP [230, 350]	108	25	20	40

autres 3GL »; mais, vu que cet ensemble ne contient que 3 projets, ses résultats ne sont pas significatifs, mais plutôt indicatifs. Ce dernier a également un PRED(0.25) de 100% pour ISBSG. Ce qui signifie que 100% des efforts prévus se retrouvent à l'intérieur de ± 25% de leurs efforts réels. Ainsi, d'après la conclusion de Conte (1986) et de Briand (1988), soit itère acceptable pour un modèle d'estimation de l'effort est PRED(0.25)>=75%, à titre indicatif, l'environnement automatisé de ISBSG est un estimateur

7.2 MODÈLE BASÉ SUR LA DROITE DE RÉGRESSION LINÉAIRE

Ce modèle basé sur la technique statistique de la régression linéaire est utilisé en considérant l'effort estimé comme variable indépendante, et l'effort réel comme la variable expliquée. Ce test est effectué pour évaluer la corrélation entre ces deux variables. Pour tous les langages, cette corrélation est positive, c'est à dire que l'effort estimé par SLIM et ISBSG, et l'effort réel varie dans le même sens, cependant l'intensité de liaison linéaire entre ces deux derniers

Tableau 7.18 : Analyse de régression linéaire entre l'effort réel et celui estimé par ISBSG et par SLIM

Types de langages	Nombre	SLIM		ISBSG	
		Équation	R^2	Équation	R^2
Autres 3GL	3	Y= 0.0995x + 1494.3	0.0084	Y= 1x – 0.0531	0.9972
Autres 4GL	31	Y= 0.061x + 11396	0.0003	Y= 1x + 0.0474	0.6183

Access	17	Y= 0.0054x + 792.52	0.0022	Y= 0.9997x + 0.2929	0.0146
Autres APG	16	Y= 4.0351x + 8469.2	0.1415	Y= 1x – 0.0249	0.3503
C	15	Y= 0.0217x + 6675.7	0.0591	Y= 1x – 0.0257	0.1903
C++	21	Y= 0.5341x + 6336.9	0.0167	Y= 1x – 0.0414	0.6252
Clipper	4	Y= 6.0106x – 2423.7	0.8535	Y= 1x + 0.0335	0.9803
Cobol	106	Y= 0.1036x + 6112.5	0.3085	Y= 1x + 0.0668	0.3666
Cobol II	21	Y= 0.7239x + 2896.9	0.7048	Y= 1x + 0.014	0.9674
CSP	7	Y= 1.2985x + 3274.1	0.4887	Y= 1x + 0.0407	0.6974
Easytrieve	8	Y= 5.9752x + 644.04	0.254	Y= 1x – 0.0157	0.195
Ideal	6	Y= -0.0064x + 13134	0.0003	Y= 1x + 0.0061	0.3844
Java (4GL)	2	Y= 24.254x + 4064.1	1	Y= 0.8152x + 1679.5	1
Java (3 GL)	2	Y= 29.603x – 4008.2	1	Y= 1.5903x – 2251.2	1
Natural	41	Y= 0.0974x + 3775.1	0.1124	Y= 1x + 0.0279	0.8577
Oracle	26	Y= 0.0712x + 5022	0.059	Y= 1x – 0.0032	0.4279
PL1	29	Y= 0.058x + 4483.1	0.0192	Y= 1x – 0.001	0.2343
Powerbuilder	18	Y= 24.961x – 2308.1	0.5036	Y= 1x + 0.0153	0.6615
SQL	20	Y=-0.0696x + 8710.6	0.0014	Y= 1x – 542.48	0.6042
SQL Forms	3	Y= 1.2043x + 1136.7	0.2063	Y= 1x + 0.0287	0.5755
SQL Windows	4	Y= 3.2928x + 6218.3	0.6403	Y= 1x + 0.0175	0.663
Telon	23	Y= 13.595x – 131.58	0.5389	Y= 1x – 0.0005	0.8555
Visual Basic	34	Y= 0.2706x + 3587.1	0.0914	Y= 1x + 0.0033	0.5612

Pour ce modèle de langages où tous les projets sont considérés, le tableau précédent montre que le coefficient de corrélation (R^2) des estimations de ISBSG est toujours plus élevé, à *Easytrieve* --qui est l'unique à avoir un coefficient de corrélation de SLIM supérieur à celui de ISBSG--, et *Java* pour qui cette valeur est égale à un (1) aussi bien pour SLIM que pour ISBSG. Ce qui est normal, car, en distinguant les projets développés en Java dans un environnement 4 GL de ceux développés dans un milieu 3 GL, il y a deux projets dans chacune de ces deux catégories, et par deux points ne passe qu'une droite et une seule. Ce qui justifie pour ce dernier langage, la corrélation parfaite positive entre l'effort estimé par ISBSG et SLIM et l'effort réel, mais, cela n'est pas significatif. Cependant tel n'est pas le cas pour les autres langages; et, de plus, la différence entre le coefficient de corrélation de ISBSG

et de celui de SLIM varie de 2.27% --avec *SQL Windows*-- à 98.88% --avec la catégorie « autres 3 GL » (tableau 7.18).

Un scénario identique se répète pour le deuxième modèle, où les langages sont catégorisés par types de génération et où les projets extrêmes par leur taille et par leur effort sont abolis (tableau 7.19).

Tableau 7.19 : Analyse de la régression linéaire entre l'effort réel et celui estimé par ISBSG et par SLIM par types de langage

Types de langage 4 GL [Taille]	Nombre	SLIM		ISBSG	
		Équation	R^2	Équation	R^2
Autres 4GL [110, 950]	12	Y= -0.308x + 3598	0.1565	Y= 1x – 0.0961	0.3543
……[951, 5000]	8	Y= 0.1912x + 11716	0.0205	Y= 1x + 0.0014	0.0861
Access [200,800]	11	Y= 0.0134x + 730.98	0.0779	Y= 0.9999x + 0.0712	0.1946
Clipper [234, 500]	3	Y= -4.8528x +7776.3	0.6484	Y= 1x - 0.0331	0.569
Java 188	1	**********	*****	**********	*****
Easytrieve [70, 200]	6	Y= 23.958x –2422.7	0.8667	Y= 1x – 0.0075	0.8662
CSP [230, 350]	5	Y= -2.0399x + 5024.3	0.2847	Y= 1x + 539.85	0.9119
Ideal [840, 3650]	3	Y= 0.1442x + 5525.2	0.6061	Y= 1x + 0.0266	0.8014
Natural [20, 620]	30	Y= -0.0154x + 2048	0.0024	Y= 1x + 0.0035	0.475
[621, 3500]	9	Y= -0.0713x + 17666	0.0755	Y= 1x – 0.0282	0.7424
Oracle [100, 2000]	19	Y= -0.0388 + 4979.9	0.0036	Y= 1x + 0.0292	0.3918
Powerbuilder [60, 480]	12	Y= 3.8323x + 1271.6	0.1084	Y= 1x – 0.0364	0.1304
SQL [280, 800]	11	Y= -0.3691x + 4003.8	0.0832	Y= 1.0001x – 0.4114	0.0005
…… [1350, 4500]	8	Y= -0.2462x + 13701	0.0313	Y= 1x - 0.0025	0.678
SQL Forms [150, 1000]	3	Y= -1.7015x + 2419.1	0.0946	Y= 1x + 0.0287	0.5755
SQL Windows [150, 600]	4	Y= 3.2928x + 6218.3	0.6403	Y= 1x + 0.0175	0.663
Visual Basic [30, 600]	24	Y= 0.0534x + 1639.2	0.0051	Y= 1x – 0.0002	0.4657

Types de langage 3 GL [Taille]	Nombre	SLIM		ISBSG	
		Équation	R^2	Équation	R^2
Autres 3GL [290, 330]	3	Y= 0.0995x + 1494.3	0.0084	Y= 1x – 0.0531	0.9972
C [200, 800]	9	Y= 0.0191x + 4126.4	0.2125	Y= 1x – 0.019	0.2908
C++ [70, 500]	12	Y= 1.0144x + 2350.6	0.2658	Y= 1 – 0.0071	0.1173
[750, 1250]	5	Y= -7.5203x + 26517	0.4297	Y= 1x – 0.1692	0.0601

Cobol [60, 400]	60	Y= 0.004x + 2401.9	0.0007	Y= 1x + 0.0065	0.4487
[401, 3500]	31	Y= 0.0694x + 14024	0.0712	Y= 1x – 0.0003	0.6462
Cobol II [80, 180]	9	Y= 10.675x – 2206.2	0.5384	Y= 1x – 0.0005	0.457
[180, 500]	6	Y= 2.2609x + 2565.2	0.0599	Y= 1x – 0.0353	0.6177
Java [150, 350]	2	**********	1	**********	1
PL1 [80, 450]	19	Y= 0.0017x + 1552.6	0.00005	Y= 1x + 0.0013	0.6441
[550, 2550]	5	Y= 0.0584x + 5013.3	0.7264	Y= 1x – 9^{E}-07	0.8699

Types de langage APG [Taille]	Nombre	SLIM		ISBSG	
		Équation	R^2	Équation	R^2
Autres APG [200, 1000]	7	Y= -0.605x + 4128.6	0.1339	Y= 1x + 0.0186	0.3535
Telon [70, 650]	18	Y= 5.6867x + 1216	0.3482	Y= 1x – 0.0132	0.7524

Pour la majorité des projets, le lien entre l'effort estimé par ISBSG et l'effort réel est plus accentué que celui entre ce même effort réel et celui estimé par SLIM, bien que l'écart soit plus ou moins grand dépendamment des langages. Cependant, le contraire se présente pour *Clipper, SQL* [280, 800] pour les langages de la quatrième génération, pour tous les projets de *C++*, et pour *Cobol II* [80, 180], en ce qui concerne les langages de la troisième génération. Ces derniers ont un coefficient de corrélation de SLIM supérieur à celui de ISBSG. Cela ne fait pas de SLIM un meilleur évaluateur pour ces langages, car, même si la corrélation avec SLIM est plus évidente qu'avec ISBSG, il reste toujours que le coefficient est loin d'être égal à 1. Le plus élevé est celui de *Clipper* et il équivaut à 0.6484.

D'un autre côté, *Easytrieve* avec ses 6 projets, a un coefficient de corrélation de SLIM (R^2= 0.8667) sensiblement pareil à celui de ISBSG (R^2 = 0.8662). Ce qui signifie que pour les deux logiciels d'estimation, l'intensité de liaison entre l'effort estimé e
presque la même. Donc, pour estimer l'effort des projets développés en *Easytrieve,* par le test de la régression linéaire, le progiciel SLIM et l'environnement automatisé de ISBSG peuvent être utilisé à égard analogue, vue la similitude de leurs résultats. Mais, cela n'est point valable pour les autres langages; car, encore une fois, une corrélation parfaite est déterminée par une valeur égale à 1 (corrélation positive) ou 1 (corrélation négative). Certes, c'est un cas extrême peu rencontré dans la pratique, mais, elle sert toutefois de point de comparaison (Baillargeon, 1984). En fait, il n'existe pas de normes absolues. En sciences

sociales, si une variable a un R^2 égal à 20%, les chercheurs sont très satisfaits d'avoir trouvé une variable qui a une valeur aussi significative pour contribuer à expliquer une partie d'une relation dans un environnement complexe, et le développement de logiciel est un environnement complexe!

Certains experts dans le domaine, comme M. Abran, considèrent qu'en estimation informatique un R^2 égal à 0.50 est déjà très significatif et qu'il s'agit d'une variable dont il faut absolument tenir compte dans un modèle d'estimation (c'est-à-dire une seule variable explique au moins 50% de la variation, et toutes les autres moins de 50%).

Tableau 7.20. : Pourcentage des langages avec un R^2 supérieur à 50%

Types de langages	SLIM (%)	ISBSG (%)
3GL	20	56
4GL	25	50
APG	0	50

Dans tous les trois types de langages (3GL, 4GL et APG), en excluant *Java*, ISBSG a toujours plus de langages qui ont un coefficient de corrélation supérieur à 50% (tableau 7.20). Il en est de même pour l'ensemble des langages qui ont une intensité de liaison linéaire entre l'effort réel et l'effort estimé par ISBSG et SLIM au moins égal à 70% (tableau 7.21).

Tableau 7.21 : Pourcentage des langages avec un R^2 supérieur à 70%

Types de langages	SLIM (%)	ISBSG (%)
3GL	11	22
4GL	6.25	25
APG	0	50

des 62% de ISBSG. D'un autre côté, seulement 10% des langages ont des efforts estimés par SLIM dont le coefficient de corrélation est supérieur à 70%. Ce qui est moindre comparé à cinq langages sur 21 pour ce qui est de ISBSG. Ce qui montre encore une fois que l'environnement automatisé de ISBSG a une meilleure qualité d'estimation par rapport à SLIM.

Somme toute, en considérant ce test de la régression linéaire, l'environnement automatisé de ISBSG est un outil acceptable pour l'estimation de l'effort des projets pour la plupart des langages, ce qui n'est pas le cas de SLIM.

De façon plus globale, les résultats ont montré que dépendamment des tests effectués pour mesurer la qualité des estimations, SLIM est plus fiable pour certains langages et ISBSG pour d'autres. Mais, une récapitulation générale permet de voir que ISBSG a une qualité d'estimation meilleure que SLIM, même s'il n'est pas très précis. Ainsi, de ce qui précède, et surtout de l'analyse de l'erreur à partir de l'effort (coût unitaire moyen), l
modèle de productivité a posteriori basé sur les lignes de code SLIM ne répond pas aux
bon 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% de ses observations* (Conte, 1986, Abran et Robillard, 1993).

avoir un impact défavorable sur le projet.

- Finalement, l'environnement de décision change avec le temps de façon autonome. Par exemple, il peut l'être à cause des modifications dans les spécifications des exigences. En décisions sur le personnel pendant une période particulière peuvent affecter la progression du projet, et par conséquent, favoriser de telles décisions pour les périodes subséquentes.

Ces problèmes sur la prise de décision, ajoutés à ceux du non respect d l'échéancier d'un projet, peuvent engendrer des conséquences néfastes pour le département d'informatique et ainsi donc, pour l'entreprise concernée. Afin de contrer cela, il a été fixé comme but pour cette recherche d'améliorer la qualité de la prise de décision d'estimation des gestionnaires des projets informatiques.

Dans la perspective d'y arriver, une revue de littérature a été effectuée. Dans cette dernière, les principes et les modèles d'estimations des coûts logiciels y sont exposés. To processus d'estimation ne s'avère pas évident pour des raisons d'incompatibilité de buts à satisfaire, de non-disponibilité d'information, de besoin de modification du code du logiciel

-à-dire l'évaluation et l'analyse de la qualité des estimations faites avec le progiciel SLIM.

Afin d'évaluer cette fiabilité de SLIM, les données de ISBSG (International Software Benchmarking Standard Group) ont été utilisées pour cette recherche, et deux critères ont été élaborés pour l'évaluation, à savoir :

- l'analyse de l'erreur, c'est-à-dire le modèle basé sur le coût unitaire moyen (effort en heure/personne);
- le modèle basé sur la droite de régression linéaire.

Cette méthodologie nous a permis d'analyser d'abord la distribution de l'échantillon de données et ensuite les résultats. Et de cette analyse découle la conclusion que la fiabilité de SLIM laisse considérablement à désirer. En effet, les estimations de ce dernier ont comparées aux efforts réels et à ceux estimés par l'environnement automatisé de ISBSG, et les résultats démontrent que pour l'ensemble des projets, la qualité des estimations de SLIM est non seulement faible, mais, elle est moindre que celle des estimations de ISBSG. Donc, dans tous les cas, l'environnement automatisé de ISBSG est un meilleur outil d'estimation que SLIM et ce, indépendamment de la taille des projets.

Cela est peut-être dû au fait que l'outil ISBSG n'a pas été développé dans le même environnement que SLIM qui lui a été développé, il y a près de trente ans à partir des données des projets reliés au département de la défense américaine (Kemerer, 1987). Ainsi, dans le

futur, pour faire de SLIM un progiciel plus fiable, il serait intéressa
l'environnement de développement du projet spécifique. Ce qui pourrait faire l'objet de travaux futurs.

De plus, la faiblesse de la précision et de la fiabilité des techniques d'estimation combinées aux risques financiers, techniques, organisationnels, et sociaux des projets logiciels, nécessitent une estimation fréquente durant le développement de l'application. Une possibilité d'amélioration des évaluations pourrait donc être d'utiliser plus d'un modèle (outil) pour estimer l'effort des projets, tel que fait dans cette recherche, avec l'environnement automatisé de ISBSG.

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 », *IEEE Transaction on Software Engineering*, 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. et al. 2000. *Software cost estimation with COCOMO II*. Englewood Cliff, New Jersey, Prentice-Hall Inc.

Boehm, B., B. Clark et al. 1995. « Cost Models for Future Software Life Cycle Processes: *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 ». *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. 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 », Logikos Inc., part I, http://www.logikos.com/projmgmt/ProjEst1.html.

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.

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

Vicinanza, S.T., M.J. Prietula et T. Mukhopadyay. 1990. « Case-based reasoning in software effort estimation». *Proceeding 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.*

Baillargeon, G. 1984. « Corrélation linéaire simple et corrélation des rangs ». Chap. in *Techniques statistiques avec applications en informatique, techniques administratives et sciences humaines*, p. 405–412. Trois-Rivières (Qué.) : Les éditions SMG.

Toffolon, C. 1998. "The Decision-Making in Software Engineering: An Econometric Model to Analyze the Determining Factors", *FESMA 98 - Business Improvement Through Software Measurement*, Antwerpen, Belgium.

Sengupta, K., Abdel-Hamid, T.K. 1996. "The impact of unreliable information on the management of software projects: A dynamic decision perspective", *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, Vol. 26, p. 177-189.

APPENDICE A

CADRE DE BASILI

<table>
<tr><th colspan="6">Définition</th></tr>
<tr><th>Motivation</th><th>Objet: sujet</th><th>Objectifs</th><th>Utilisateurs</th><th>Domaine</th><th>Étendue</th></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>Analyser et évaluer la qualité des estimations</td><td>Gestionnaires en informatique (chefs de projet, décideurs)

Praticiens qui estiment les coûts en développement de logiciels</td><td>Projets de développement de logiciels de type MIS</td><td>Base internationale de données ISBSG (1999)

Modèle d’estimation à posteriori</td></tr>
<tr><th colspan="6">Planification</th></tr>
<tr><th colspan="2">Conception</th><th colspan="2">Critères</th><th colspan="2">Sélection de mesure</th></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)
• Deux types d’échantillon : complets et avec élimination des «outliers»

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</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>

	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	
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 • Base de donnée épurée : échantillon
Interprétation		
Contexte d'interprétation	**Extrapolation**	**Travaux futurs**
• De tout le progiciel SLIM, seule la composante SLIM-estimate a été utilisée. • Contexte statistique : o 457 projets o 22 langages o 3 types de langages • Les objectifs ont été atteints • Cette recherche pourrait contribuer à mieux évaluer les modèles de productivité à posteriori dans le développement de projet logiciel	Positifs : • L'échantillon est de qualité par sa taille et par sa composition • Deux types de test ont été utilisés Négatifs : • Élimination de certains langages de l'échantillon, car, ils n'y avaient pas assez de projets	Calibrer SLIM à l'environnement de développement d'un projet spécifique

APPENDICE B

FACTEUR D'ÉQUIVALENCE DU PROGICIEL SLIM

SIZING OPTIONS

The purpose of the sizing options dialog box is to allow you to customize the sizing parameters to your development environment.

FUNCTION UNIT

This is the area where you have the opportunity to specify the sizing units currently in use within your organization. When you press the list box arrow, a list of possible sizing units is presented. Point and click on the one that you plan to use. If you are not able to find the appropriate size measure, please contact QSM and request that your size metric be included in any future release.

GEARING FACTOR

Because on e of the essential inputs to the SLIM model is the size of the system in ESLOC, it is necessary to define the relationship between your chosen function unit and ESLOC. The gearing factor defines this relationship. If you size the system in ESLOC, this factor is almost is almost always 1.0. However, if you typically size your systems in some other unit, you should attempt to calibrate this gearing factor from your own historical data. In the absence of any data, you may want to use the following table as a guideline for selecting the gearing factor for the various function units available in SLIM. Note that the range is rather broad in most cases. This is because of the wide variation between languages as well as development styles and even the meaning of some of these terms between organizations. In general, however, you may want to use a value close to the lower end of the range for fourth

and fifth generation languages, while the upper end of the range is more appropriate for lower level languages.

Function Unit	**ESLOC/Funit Range**
ESLOC	1 - 1
Function Points	5-500
Feature Points	5-500
Subsystems	5000-75000
Programs	100-10000
Modules	100-5000
Objects	100-500
Entities	100-10000
Processes	500-1000
CSU	5-200
CSCI	5000-75000
CSC	100-100000
Mark II Function Points	5-500
Method II Function Points	5-500
Dialog Boxes	10-1500
Actions	100-500
Screens	10-5000
Reports	10-2000
Windows	1-5000

LANGUAGES LEVEL TABLE FOR FUNCTION POINTS

Because the most commonly used function unit after ESLOC is one of the Function Point metrics and more data exists for this unit, we have provided the following mappings for function points by language level.

Language Level	**Function Pt Language Factor**
Low level	200-400
2nd Generation	75-200
3rd Generation	30-75
4th Generation	15-30
5th Generation	5-15

FUNCTION POINT LANGUAGE TABLE

As a further breakdown of function point language factors, you may refer to the following widely published table which show many of the function point language factors broken down by specific language.

Language	**Function Point Language Factor**
Excel, QUATTRO PRO, Mosaic, Screen painters, Spreadsheet languages	5
Database Query, PACBASE, SQL, PowerBuilder, TI-IEF, Program Generators, TELON, Notes, Nexpert, Java, Fusion, Delphi	15
ADS/Online, MUMPS, NETRON/CAP, SMALLTALK, Oracle Forms, Report Writer, Easytrieve, SP	20
APL, Objective C, Visual Basic	30
MS C++ V7, CLIPPER, FOXPRO, Object-oriented default, Statistical default, Decision support default	35
Database Languages, Al Shells, FOCUS, ORACLE, RAMIS II, SYBASE	40
Informix, Simulation default, C++	45
English Based Language, MAPPER, Natural, RPG III	55
Ada, QuickBaasic, RPG II	60
BASIC, ,PROLOG, Object Assembly	65
PL/1, Ada 83, Problem-oriented default, TALON, PLM	70
REX II, 3rd Generation Default	80
PASCAL, ANSI COBOL 85, Tandem (TAL)	90
ALGOL 68, CHILL, FORTRAN, UNIX, Shell scripts, JOVIAL	105
C Default	130
Macro Assembler, JCL	220
Basic Assembly, SPS	320
Machine Language	640

APPENDICE C

LANGAGE *NATURAL* : COMPORTEMENT DES ESTIMATIONS DE SLIM AVEC LA VARIATION DU FACTEUR D'ÉQUIVALENCE

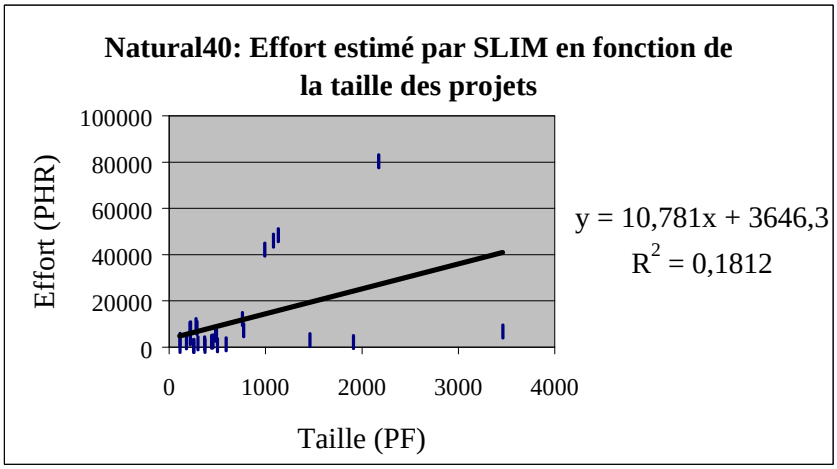
L'unité principale de l'outil d'évaluation SLIM est les lignes de code. Et, étant donné que la taille des projets de la base de données ISBSG est exprimée en points de fonction, il a été nécessaire de la convertir en lignes de code en utilisant le tableau pré-établi des facteurs d'équivalence de l'appendice B. Ce dernier figure dans la documentation de SLIM, cependant l'origine de ces facteurs de conversion n'y est pas identifiée. En poursuivant *Natural*, ce tableau montre que *Natural* a un facteur d'équivalence de 55. Et tous les résultats des chapitres précédents dépendent de cette valeur. -il de ces résultats si le facteur de conversion était de ± 55, c'est-à-dire 40 ou 70?

La réponse à cette question fait l'objet des parties subséquentes. Mais tout d'abord, il faut noter que le fait de changer le facteur d'équivalence réduit la taille de l'échantillon de *Natural*. Avec un facteur d'équivalence de 40, certains projets une fois convertis, ont une taille inférieure à 1000 lignes de code, qui est le minimum qu'un projet doit avoir pour pouvoir être estimé par SLIM. Et, d'autre part, il y a des projets qui présentent un effectif inférieur à 0.5 personne qui également le plancher considéré par SLIM. Ainsi, tous les projets qui rencontrent ces limites sont éliminés; ce qui fait que l'échantillon de *Natural* passe de 41 à 28 projets. Cependant, cela n'est pas le cas de l'échantillon de *Natural* lorsque le facteur de conversion 70 est considéré. Donc, pour ce dernier, l'ensemble des 41 projets de l'échantillon

I. Modèle avec tous les projets de *Natural* (avec les points extrêmes)

Tableau 1 : Comportement des estimations de SLIM avec un facteur d'équivalence de 40

Portion de SLIM sous-estimée par rapport à l'effort réel	54%
Portion des estimations de ISBSG inférieures à celle de SLIM	57%
MMRE	326.30%
RRMS	303%
PRED (0.25)	7%



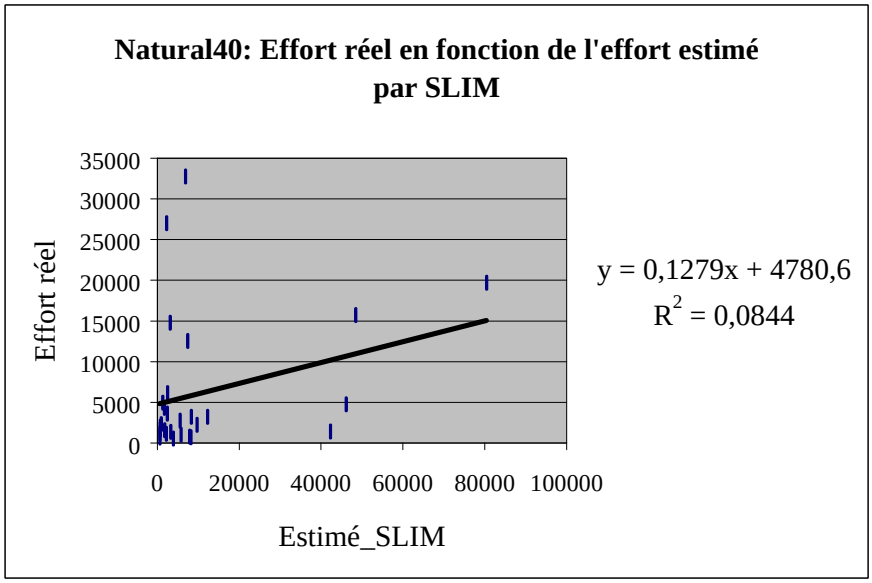
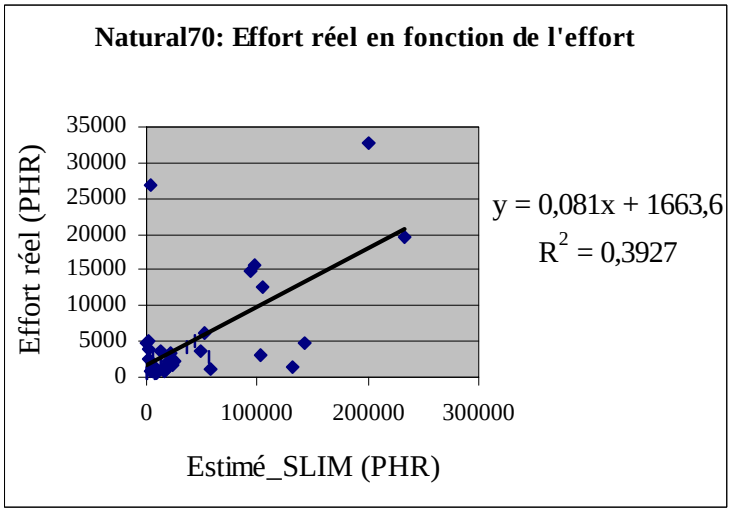
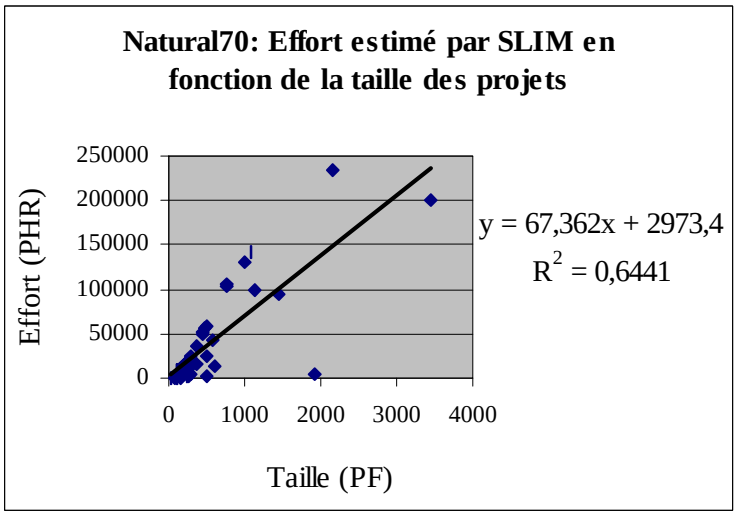


Tableau 2 : Comportement des estimations de SLIM avec un facteur d'équivalence de 70

Portion de SLIM sous-estimée par rapport à l'effort réel	19.5%
Portion des estimations de ISBSG inférieures à celle de SLIM	90%
MMRE	1168%
RRMS	1257%
PRED (0.25)	5%





À l'image de ces illustrations, il est évident que le facteur d'équivalence a une influence sur les estimations faites par l'outil SLIM. Mais, il est intéressant de voir la nature de ces estimations avec les modèles développés sans les points extrêmes.

II. Modèles développés des projets de *Natural* (sans les points extrêmes)

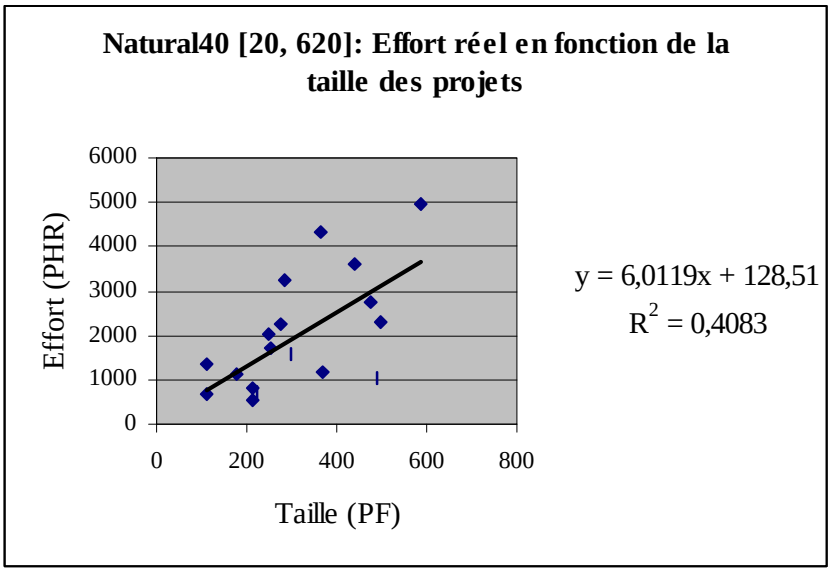
Une fois les points extrêmes éliminés, l'échantillon de *Natural* se présente comme suit :

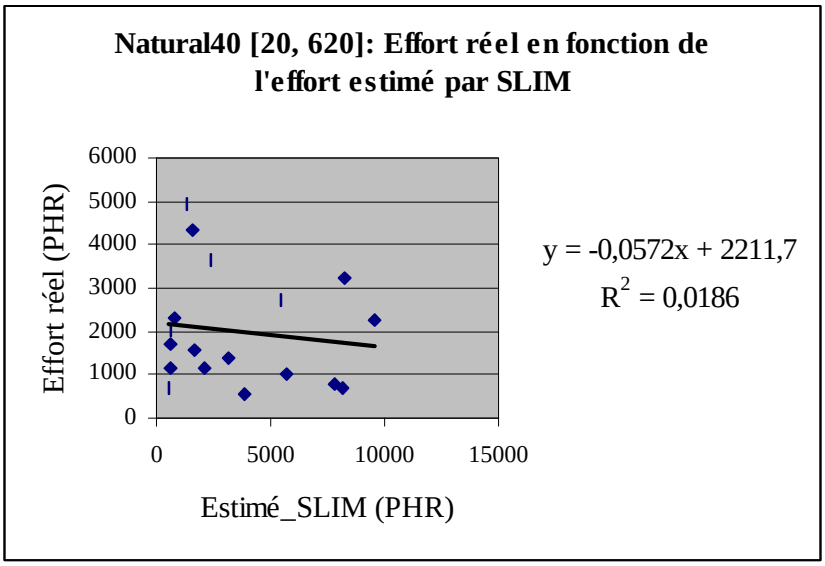
- Pour le facteur de conversion 40 : - *Natural [20, 620]* comprend 18 projets
  - *Natural [621, 3500]* comprend 9 projets
- Pour le facteur de conversion 70 : - *Natural [20, 620]* contient 29 projets
  - *Natural [621, 3500]* contient 9 projets

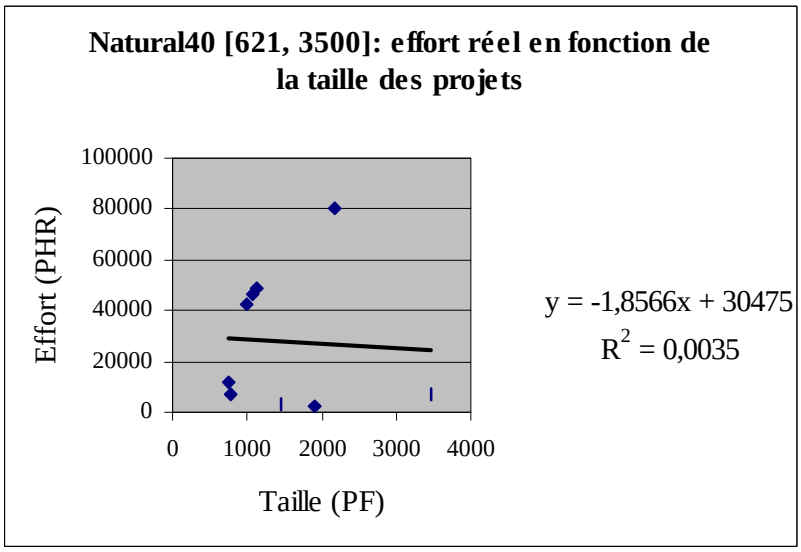
Il est à noter que toutes les statistiques et graphiques suivants sont en fonction de ses tailles respectives de l'échantillon.

Tableau 3 : Comportement des estimations de SLIM avec un facteur d'équivalence de 40

	Natural [20, 620]	Natural [621, 3500]
Portion de SLIM sous-estimée par rapport à l'effort réel	56%	56%
Portion des estimations de ISBSG inférieures à celle de SLIM	50%	67%
MMRE	235.17%	538.11%
RRMS	188%	223%
PRED (0.25)	11%	0%




Natural40 [20, 620]: Effort réel en fonction de l'effort estimé par SLIM
Effort réel (PHR)
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
Estimé_SLIM (PHR)
y = -0,0572x + 2211,7
R2 = 0,0186


Natural40 [621, 3500]: effort réel en fonction de la taille des projets
Effort (PHR)
100000
80000
60000
40000
20000
0
0
1000
2000
3000
4000
Taille (PF)
y = -1,8566x + 30475
R2 = 0,0035

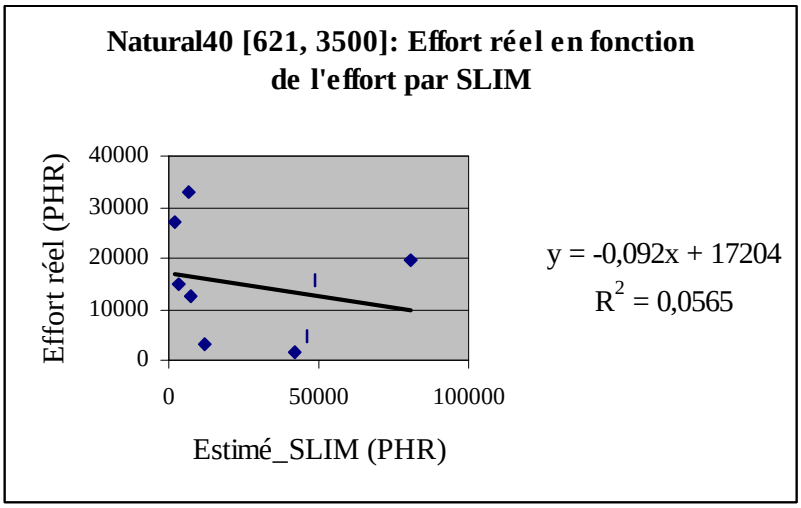
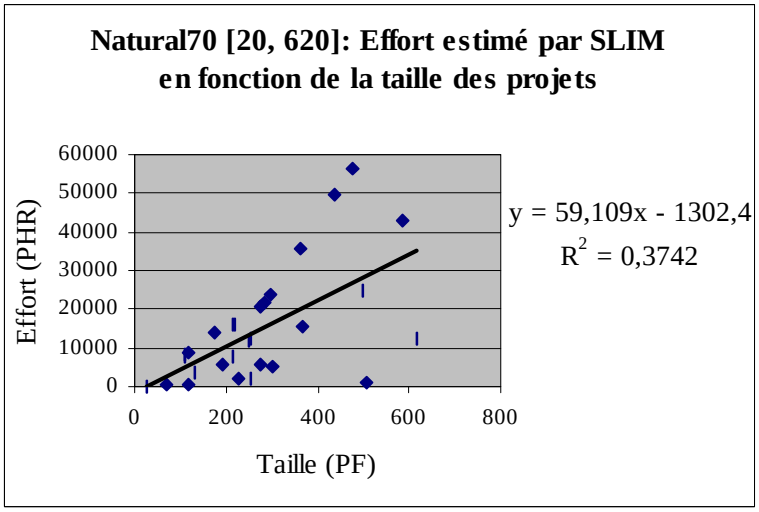
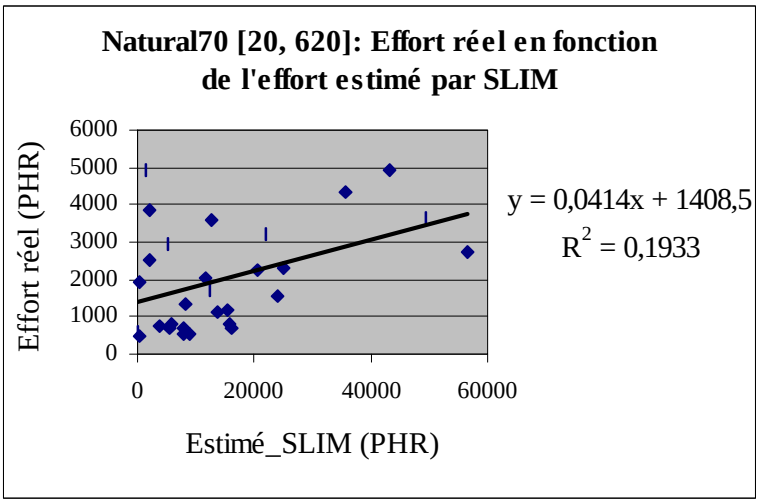
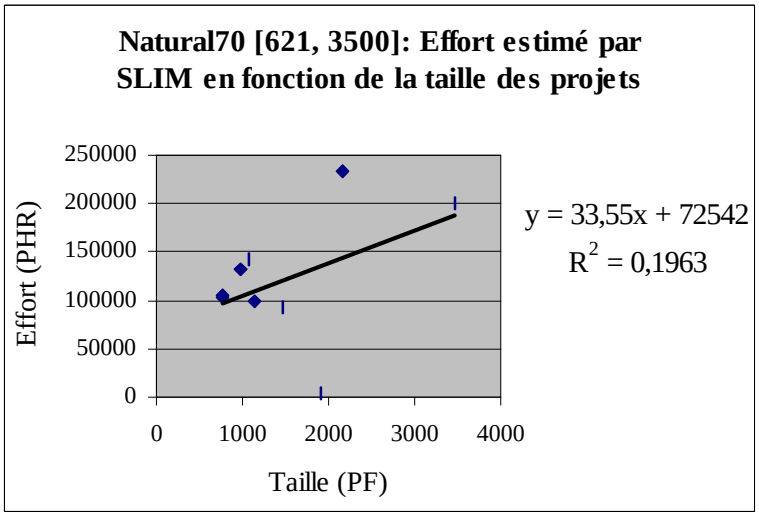


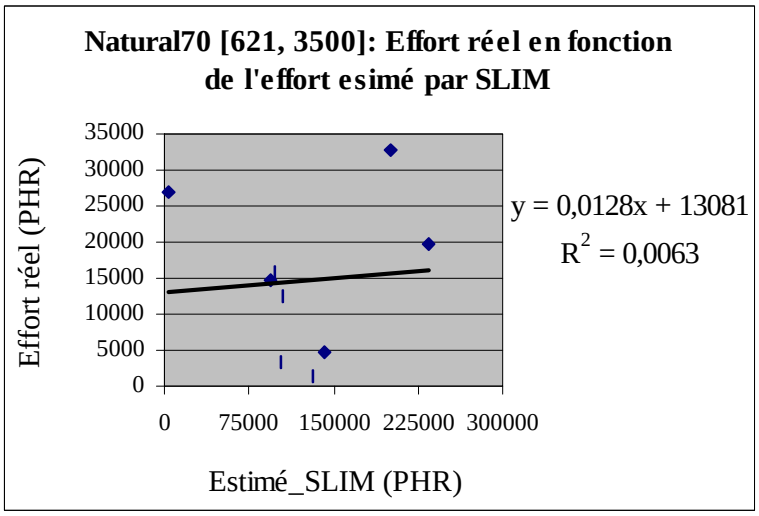
Tableau 3 : Comportement des estimations de SLIM avec un facteur d'équivalence de 70

	Natural [20, 620]	Natural [621, 3500]
Portion de SLIM sous-estimée par rapport à l'effort réel	21%	11%
Portion des estimations de ISBSG inférieures à celle de SLIM	86%	89%
MMRE	784.19%	2073.47%
RRMS	941%	857%
PRED (0.25)	7%	0%




Natural70 [20, 620]: Effort réel en fonction de l'effort estimé par SLIM
Effort réel (PHR)
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
Estimé_SLIM (PHR)
y = 0,0414x + 1408,5
R2 = 0,1933


Natural70 [621, 3500]: Effort estimé par SLIM en fonction de la taille des projets
Effort (PHR)
250000
200000
150000
100000
50000
0
0
1000
2000
3000
4000
Taille (PF)
y = 33,55x + 72542
R2 = 0,1963



Ces tableaux et figurent montrent sans nul doute que les résultats des estimations de SLIM changent dès que le facteur de conversion est modifié. Le but de cette partie étant de montrer qu'une éventuelle modification de ce facteur d'équivalence se reflète sur les évaluations faites par SLIM, nous ne nous attarderons pas sur l'ampleur de cette influence.

APPENDICE D

GRAPHIQUE DE COMPARAISON DE L'EFFORT RÉEL À L'EFFORT ESTIMÉ PAR SLIM ET ISBSG