Mesure de la fiabilité de SLIM (Software Life-Cycle Model)

Iphigénie Ndiaye, Alain Abran, Ghislain Lévesque
Laboratoire de recherche en gestion des logiciels
Université du Québec à Montréal

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 error of ± 25% for 75% of cases (Conte, 1986; Verner, 1992, Abran et Robillard, 1993).

Keywords: Estimation, Measurement, SLIM, ISBSG, Software Projects

1. Introduction

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.

Cet article présente une évaluation de la qualité des estimations faites par le progiciel d'estimation SLIM. La première partie décrit l'outil d'estimation SLIM dans son ensemble, et la deuxième l'échantillon de données. À celle-ci suit la méthodologie, partie qui présente la stratégie utilisée pour mesurer la qualité des estimations du progiciel SLIM. La quatrième partie quant à elle, est consacrée 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. Finalement, 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.

2. Description sommaire de l'outil SLIM

SLIM est un modèle d'estimation développé par Putnam en 1978. Il est l'un des premiers modèles algorithmiques d'estimation de coût. Il est généralement connu comme un « macro-modèle » d'estimation c'est-à-dire qu'il est approprié aux gros projets. Putnam utilise la courbe de Rayleigh conjointement avec un nombre d'hypothèses dérivées empiriquement pour obtenir l'équation suivante (Kitchenham et Taylor, 1984; Putnam et Myers, 1997) :

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

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

Le modèle SLIM comprend trois composantes qui sont l'estimation, la mesure et le contrôle[1]. Ces trois outils répondent à des objectifs différents :

- SLIM-Estimate : est utilisé pour développer un plan de projet, pour un ensemble de circonstances données.
- SLIM_Metrics : est utilisé pour déterminer la performance, la position concurrentielle et les tendances du développement.
- SLIM-Control : est utilisé pour s'assurer que les projets rencontrent les attentes et pour faire des corrections tactiques en cours de projet.

3. Description des données

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

Une alternative est apparue récemment avec la disponibilité d'une base de données provenant de ISBSG (*International Software Benchmarking Standards Group)*. Ce groupe s'intéresse à la collecte, la validation et la publication d'une base de données historiques des projets de développement logiciel. 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 complété entre 1989 et 1998.

3.1 Analyse de l'échantillon

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

3.2 Critères de base

Parmi les 789 projets de l'entrepôt de ISBSG-1999, seuls ceux qui répondent aux caractéristiques suivantes seront pris en considération (Oligny 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 été choisi sur la base que les

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

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 Oligny, Bourque et Abran (1997) dans le domaine). Seuls 497 projets rencontrent ces critères et une analyse statistique de l'échantillon est décrite comme suit :

Tableau 1 : Analyse statistique descriptive de l'échantillon

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

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

Tableau 2: Projets par langages de programmation

Langages de programmation	Nombre de projets
Access	17
Assembler	2
C	15
C++	21
Clipper	4
Cobol	106
Cobol II	21
CSP	7
Easytrieve	8
Focus	2
Ideal	6
Ingres	1
Java	4
Natural	41
2GL	1
Inconnus	21

Langage de programmation	Nombre de projets
Oracle	26
Oracle Forms	2
Oracle SQL	2
Other 3GL	3
Other 4GL	31
Other APG	16
Pascal	2
PL/I	29
Powerbuilder	18
SQL	20
SQL Forms	3
SQL Windows	4
Telon	23
Unix Script	1
Visual Basic	34

Parmi ces 497 projets, 21 n'ont pas d'identification des langages de programmation utilisés et du facteur d'équivalence (Appendice) qui permet la conversion des lignes de codes en points de fonction, qui est l'unité principale de la taille des différents projets de ISBSG. Il s'agit des projets logiciels développés en Sheer, Rally, PL/SQL, HPS, Drift et Abap.

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

4. Méthodologie

Suite à l'épuration des données de ISBSG décrite ci-dessus, l'évaluation de la qualité des estimations de SLIM avec ISBSG a été entamée. Cette dernière a été effectuée en introduisant la taille, la durée et le langage des projets de ISBSG comme paramètres dans SLIM_Estimate, et de là faire la comparaison entre l'effort réel des projets respectifs de la base de données ISBSG et la résultante des estimations de SLIM. Pour cette comparaison, deux sélections de mesure ont été utilisées, à savoir, l'analyse de l'erreur et la régression linéaire. Pour l'analyse de l'erreur, les formules suivantes ont été utilisées (Conte et al, 1986) :

L'erreur relative majeure : $MRE = |RE| = \left| \frac{Eact - Eest}{Eact} \right|$

La moyenne de l'erreur relative majeure :

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

L'erreur relative moyenne :

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

La racine carrée de l'erreur relative moyenne :

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

Le niveau de prédiction : $PRED(l) = \frac{k}{n}$ où k équivaut au nombre de projets logiciels dans un ensemble particuliers de données n projets dont MRE =< k/n.

Pour qu'un modèle soit acceptable, il faut que soit la MMRE soit inférieure ou égale à 0.25, soit le PRED(0.25)>=0.75. D'autre part, plus $\overline{RMS}$ ou la RMS est faible, meilleure est la prédiction.

La deuxième méthode de mesure est d'utiliser le modèle basé sur la droite de régression linéaire. Il s'agit là de mesurer la corrélation entre l'effort estimé et l'effort actuel. La régression a été faite en utilisant l'effort actuel comme variable dépendante et l'effort estimé comme variable explicative (Kemerer, 1987; Abran et Robillard, 1993, 1996).

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

3.3 Analyse de la distribution de l'échantillon de données

Afin d'avoir des résultats de comparaison crédibles, les données ont été analysées. En fait, il s'agissait d'étudier pour un même langage de programmation, la dispersion des projets et de veiller à ce que l'échantillon de données contiennent des projets de même nature, étant donné que la mesure de la qualité des estimations de SLIM nécessite un échantillon homogène. Pour cela, les 470 projets disponibles de la base de données ISBSG ont été étudiés, résultant, pour chaque langage, au développement d'un ou de deux modèles d'échantillon qui seront utilisés afin de poursuivre l'étude. Le langage *Natural* a été utilisé comme exemple et la distribution de tous ses 41 projets est présentée comme suit :

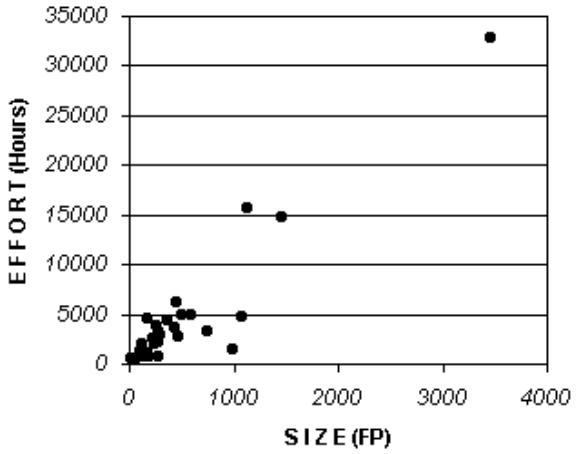

Figure 1 : Langage *Natural* dans son ensemble

Cette figure est représentée par l'équation de la droite suivante : Y = 9.0955X – 142.58 avec un coefficient de corrélation R^2 = 0.8679.

Cela indique que la taille d'un projet explique très bien l'effort fourni lors du développement de ce dernier; mais, il y a des points extrêmes comme à 3500 points de fonction qui ont trop d'influence sur la droite de régression. Alors, le modèle de régression représente mal les petits projets.

Elle montre également que l'échantillon contient plus de projets dont la taille est petite ou moyenne que ceux dont la taille est grande. Sans doute, de l'analyse visuelle de ce graphique ressortent deux sous-ensembles :

- Un premier relativement homogène pour les projets de taille inférieure à 600 points de fonction.
- Un deuxième sous-ensemble avec un peu plus de dispersion et une pente plus forte et différente du premier sous-ensemble pour les projets de plus grande taille supérieure à 800 points de fonction.

Prises telles quelles, de pareilles distributions de données représentent un faible support pour une seule régression linéaire pour les deux sous-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 3.2 et 3.3 suivantes :

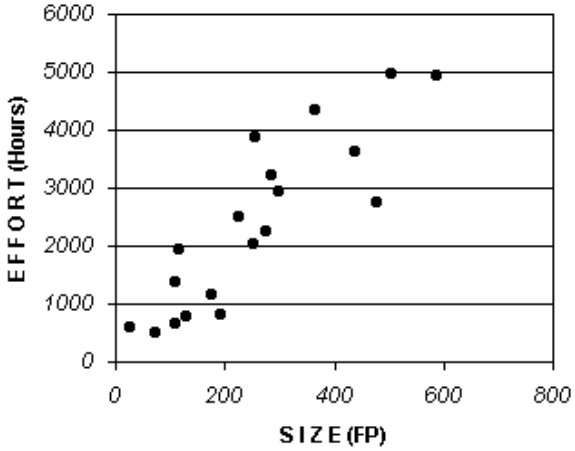

Figure 2 : Langage *Natural* et ses projets de taille inférieures à 620 points de fonction

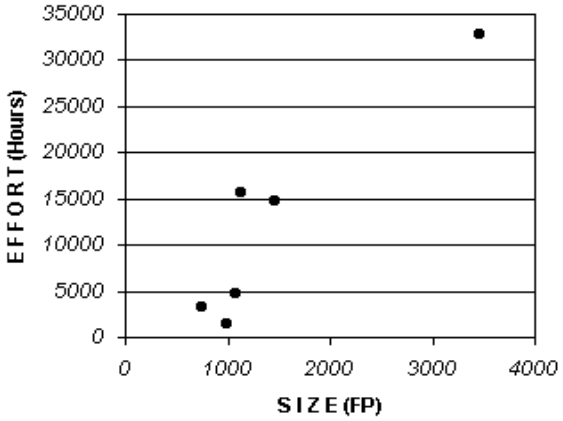

Figure 3 : Langage *Natural* et ses projets de taille supérieures à 620 points de fonction

Cette figure 2 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 ($Y = 8.1908X + 177$ et $R^2 = 0.705$). 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 3.3 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 $Y = 10.905X - 4007.8$ et un coefficient de corrélation de 85.12%.

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 exp lique 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 3 : 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

Projets de ISBSG avec les points extrêmes				
Langage	Nombre	Taille en points de fonction	Équation	R^2
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

Le tableau 3 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'objet d'un autre modèle et afin d'éviter des biais et dans certains cas, les points extrêmes sont éliminés, comme présenté dans le tableau 4.

Tableau 4 : 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	20-620	Y= 6.1355x + 264.98	0.475
	9	620-3500	Y= 10.539x – 1404.9	0.7424
Oracle	19	100-2000	Y= 7.7803x – 1280.7	0.3918
PL/1	19	80-450	Y= 8.3264x – 197.67	0.6441
	5	451-2550	Y= 5.4914x – 65.349	0.8699
Powerbuilder	12	60-400	Y= 1.9942x + 1560.1	0.1304
SQL	11	280-800	Y= -0.4274x + 3831.2	0.0005
	8	801-4500	Y= 9.2394x – 6064.3	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

4. RÉSULTATS

Natural est toujours utilisé comme exemple. 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. Ces langages sont de trois types différents, à savoir, des langages procéduraux de la 3e génération, des outils de la 4e 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% de ses projets sont développés en 3GL et les 50% restants sont développés dans un environnement de 4GL (exemple tableau 4.5).

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. 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. 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 un processus « boîte noire ».

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

4.1 Analyse des résultats pour le modèle initial avec les points extrêmes

Tout d'abord, afin que la comparaison soit menée à bien avec le modèle initial où tous les projets sont considérés, 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.

En comparant les évaluations de ISBSG à celles faites par SLIM, pour le modèle initial avec les points extrêmes, la plupart langages 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 celles 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, pour ce même modèle initial, bien vrai que l'environnement automatisé de ISBSG paraît plus fiable, c'est-à-dire que ses estimations se rapprochent plus de la réalité, les erreurs restent quand même hautes pour la majorité des langages. 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 les points extrêmes. Ce qui introduit la deuxième partie de l'analyse.

4.2 Analyse des résultats pour les modèles développés sans les points extrêmes

4.2.1 Comparaison de l'effort réel à l'effort estimé par SLIM

Comme cela a été décrit dans la partie 3.4, 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.

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 qui avait donné pour *Natural* une sous-estimation de 60% pour son premier modèle (Natural [20, 620][2]), et une surestimation de 56 % pour son ensemble de 9 projets (Natural [621, 3500]). 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. De façon plus générale, les deux modèles du langage *Natural* ont des estimations de nature distincte, à savoir une surestimation de ceux dont la taille est comprise entre 621 et 3500 points de fonction et une sous-estimation des efforts des projets du premier ensemble de 30 projets.

Cette sous-évaluation représente ainsi la nature des estimations des efforts des trois types de langages utilisés, à savoir, les langages de troisième génération, ceux de la quatrième et finalement les outils des générateurs d'applications. Un comportement identique a été remarqué pour ces mêmes langages, mais cette fois-ci dans le modèle avec les points

[2] Natural [20, 620] : ensemble des projets développés en *Natural* et ayant une taille comprise entre 20 et 620 points de fonction.

extrêmes de la partie précédente. En effet, le fait d'éliminer les points qui étaient extrêmes par leur grande taille et par leur effort élevé, n'a pas vraiment influencé l'état de l'estimation, car les projets présentent des efforts en général sous-estimés, même si les pourcentages de sous-évaluation diffèrent d'un modèle à un autre.

De l'analyse détaillée des données dans Ndiaye (2001), l'on peut déduire que le progiciel SLIM fournit en général des efforts estimés très souvent inférieurs aux efforts réels. Les erreurs relatives moyennes sont très 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 (1987). Ainsi, d'après l'étude de 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, COCOMO, ESTIMACS), il a collecté les 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.

4.2.2 Comparaison de l'effort réel à l'effort estimé par ISBSG

Avec le modèle construit directement avec le données ISBSG, 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%. Et dans l'ensemble, 59% 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.

4.2.3 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 avec des MMRE inférieures à celles de SLIM. C'est aussi le cas de 9 des 11 des langages de programmation de niveau 3 GL.

Tableau 5 : 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
PL1 [550, 2550]	5	249.32	28.80	ISBSG<SLIM

Types de langage 4GL	Nombre de projets	Erreur relative moyenne (MMRE)		Conclusion
		SLIM (%)	ISBSG (%)	
Autres 4GL [110, 950]	8	22,33	40.88	ISBSG>SLIM
Autres 4GL [951, 5000]	12	69,46	87.92	ISBSG>SLIM
Access [200, 800]	11	147,04	13.50	ISBSG<SLIM
Clipper [234, 500]	3	92,27	124.99	ISBSG>SLIM
Java 188	1	91,12	36.30	ISBSG<SLIM
Easytrieve [70, 200]	6	87,72	21.93	ISBSG<SLIM
CSP [230, 350]	5	63,28	32.33	ISBSG<SLIM
Ideal [840, 3650]	3	133,65	29.67	ISBSG<SLIM
Natural [20, 620]	30	248,01	57.86	ISBSG<SLIM

Types de langage 4GL	Nombre de projets	Erreur relative moyenne (MMRE)		Conclusion
		SLIM (%)	ISBSG (%)	
Natural [621, 3500]	9	850,63	98.14	ISBSG<SLIM
Oracle [100, 2000]	19	596,98	85.78	ISBSG<SLIM
Powerbuilder [60, 480]	12	89,52	27.78	ISBSG<SLIM
SQL [280, 800]	11	143,45	195.99	ISBSG>SLIM
SQL [1350, 4500]	8	109,54	124.72	ISBSG>SLIM
SQL Forms [150, 1000]	3	67,51	44.58	ISBSG<SLIM
SQL Windows [150, 600]	4	95,04	9.19	ISBSG<SLIM
Visual Basic [30, 600]	24	123,57	61.45	ISBSG<SLIM

Types de langage APG	Nombre de projets	Erreur relative moyenne (MMRE)		Conclusion
		SLIM (%)	ISBSG (%)	
APG [200, 1000]	7	65,21	51.01	ISBSG<SLIM
Telon [70, 650]	18	90,38	25.21	ISBSG<SLIM

Pour les autres langages de programmation, la comparaison de leur erreur relative moyenne (tableau 5), montre que 23 ensembles de projets sur 30 des trois types de langages ont des estimations de l'effort plus précises avec ISBSG qu'avec SLIM, et la plupart du temps ISBSG donne une amélioration considérable. Certes, elles ne sont pas exactes, mais, elles sont plus proches de l'effort fourni en réalité.

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 leur erreur relative moyenne est inférieure à 25% : il s'agit tout d'abord de *Access* et de *Telon*. Ensuite, 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 ne sont des modèles acceptables pour l'estimation des efforts des projets. Cela s'explique par certains pourcentages trop élevés du tableau 4.5. 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 d'évaluation 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 d'estimation.

4.3 Analyse de l'erreur à partir du $\overline{\text{RMS}}$ et du (PRED(0,25)

Afin de s'assurer de la validité des résultats des deux parties précédentes, d'autres tests sont effectués dans celui-ci, sur les mêmes données de ISBSG compte tenu des deux modèles, à savoir celui qui contient la totalité des projets et l'autre qui composé de plusieurs ensembles de projets développés à l'aide de la taille de chacun d'eux. Il s'agit du test basé sur la régression linéaire. La racine carrée de l'erreur relative moyenne ($\overline{\text{RMS}}$ ou RRMS) et le niveau de prédiction à 25% (PRED(0,25)) sont aussi calculés pour tous les projets.

En ce qui concerne le modèle initial avec les points extrêmes, pour l'ensemble des langages, la qualité meilleure des estimations de ISBSG par rapport à celles de SLIM ne passe pas inaperçue: 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 le PRED(0,25) de SLIM est plus faible que celui de ISBSG pour la majorité des langages (21 sur 23).

Et pour ce qui est des modèles développés sans les points extrêmes, les tableaux qui suivent montrent les résultats en pourcentage du $\overline{\text{RMS}}$ et du PRED(0,25).

Tableau 6 : 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
Ideal [840, 3650]	248	21	0	33
Natural [20, 620]	243	50	10	27
[621, 3500]	347	35	11	33
Oracle [100, 2000]	319	120	0	21
Powerbuilder [60, 480]	95	29	0	58
SQL [280, 800]	136	81	0	27
[1350, 4500]	127	45	0	25
SQL Forms [150, 1000]	94	30	0	33
SQL Windows [150, 600]	96	10	0	100
Visual Basic [30, 600]	122	54	0	42

Types de langage 3 GL [Taille]	RRMS		PRED (0,25)	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)
Autres 3GL [290, 330]	65	2	67	100
C [200, 800]	1653	50	11	22
C++ [70, 500]	97	86	8	25
[750, 1250]	95	24	0	60
Cobol [60, 400]	400	42	7	43
[401, 3500]	348	51	16	35
Cobol II [80, 180]	89	29	0	78
[180, 500]	109	46	17	33
Java [150, 350]	87	29	0	50
PL1 [80, 450]	274	45	5	42
[550, 2550]	895	21	20	60

Types de langage APG [Taille]	RRMS		PRED (0,25)	
	SLIM (%)	ISBSG(%)	SLIM (%)	ISBSG (%)
Autres APG [200, 1000]	97	48	14	71
Telon [70, 650]	100	22	0	56

Les comportements des deux outils d'estimation sont restés tels quels; c'est-à-dire qu'en ce qui concerne les deux critères d'évaluation ($\overline{RMS}$, et PRED(0,25)), les estimations de l'environnement automatisé de ISBSG sont de meilleure qualité que celles de SLIM. Cependant, ces deux outils ne sont pas aptes à prévoir la performance réelle des projets, vu que la racine carrée de l'erreur relative moyenne reste quand même élevée, sauf pour la catégorie « *autres 3GL* »; toutefois, 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), soit disant qu'un critère 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 admissible pour les projets développés avec les autres langages 3GL. Il l'est également pour *Access* [200, 800], pour *Easytrieve* [70, 200] et pour *SQL Windows* [150, 600] qui ont respectivement un PRED(0,25) de 91%, de 83 % et de 100%. Par contre, pour ce qui est du reste des langages, peu importe la catégorie à laquelle ils appartiennent (3GL, 4GL ou APG), l'environnement automatisé de ISBSG n'est pas un bon modèle d'évaluation. De plus, étant donné que les $\overline{RMS}$ de SLIM sont trop élevés et que ses

PRED(0,25) trop faibles par rapport à ceux de ISBSG, par conséquent, SLIM n'est non plus un bon estimateur pour l'ensemble des langages.

Ces conclusions confirmant les résultats des parties précédentes, ont été tirées grâce à une analyse de l'erreur des estimations à partir du modèle basé sur le coût unitaire moyen représenté par l'effort (heure/personne). Ces résultats peuvent également être vérifiés d'une autre façon, soit par le modèle basé sur la droite de régression linéaire. Et c'est ce qui fait l'objet de la partie suivante.

4.4 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 varie d'un langage à l'autre (tableau 7).

Tableau 7: 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]	Nb	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]	Nb	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]	Nb	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.

En outre, 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 d'un logiciel est un environnement complexe!

Tableau 8. : 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 4.8). 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 9).

Tableau 9 : 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

De plus, en ce qui concerne les modèles avec les points extrêmes, 62% des langages ont des efforts estimés par ISBSG qui expliquent au moins 50% de la variation, contre 24% pour les efforts estimés par SLIM qui sont également une partie intégrante 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.

5. Conclusion

En conclusion, 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'on déduit que le modèle de productivité a posteriori basé sur les lignes de code SLIM ne répond pas aux critères des «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, 1996).

Cette recherche peut être bénéfique aux praticiens qui utilisent les progiciels pour estimer les coûts en développement de logiciel : il s'agit surtout de leur permettre de mieux apprécier l'importance d'une estimation exacte, étant donné les conséquences néfastes que l'inexactitude des estimations pourraient entraîner dans l'entreprise. Il pourront ainsi éviter le pire pour l'entreprise concernée, en mesurant au préalable la fiabilité de l'outil d'estimation dont ils disposent et cela en utilisant la méthodologie de cette recherche. Cela engendrerait probablement des coûts supplémentaires, mais, cela n'est-il pas mieux que de prendre le risque d'avoir des surestimations ou des sous-estimations aberrantes, et ainsi, réduire la productivité moyenne, l'efficience et l'efficacité de la firme ?

Pour ce qui est des chefs de projets et des décideurs, cette recherche leur est principalement utile dans le sens où, la précision de l'estimation de l'effort de développement d'un projet logiciel facilite leur prise de décision, tâche qui n'est point facile et qui revêt un caractère crucial pour l'entreprise.

Les résultats de cette recherche sont probablement dus 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 intéressant de le calibrer d'abord à l'environnement de développement du projet spécifique.

SLIM doit être adapté au contexte de chaque entreprise en ajustant les facteurs d'équivalence (appendice) aux projets de cette même entreprise. 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. Ce dernier figure dans la documentation de SLIM, cependant l'origine de ces facteurs de conversion n'y est pas identifiée. De plus, après des tests effectués sur tous les langages de la base de données ISBSG, les résultats des estimations de SLIM changent dès que le facteur de conversion est modifié.

Ainsi, ces facteurs de conversion sont sans nul doute une raison qui pourrait fausser les estimations faites avec SLIM, vu qu'il a été développé avec des projets de la défense américaine; ainsi, une façon de pouvoir réduire les coûts inutiles d'une estimation inexacte est, pour un projets de type MIS, de développer des facteurs d'équivalence adéquats au profil de l'entreprise, à partir d'une base de données de ses projets historiques, si jamais, elle est disponible.

Ce dernier point ne concerne pas seulement les firmes qui utilisent l'outil SLIM, mais est valable pour toute entreprise qui utilise un modèle d'évaluation a posteriori développé dans un environnement autre que celui de l'entreprise concernée.

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

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.

Abran, A. et P.N. Robillard. 1996. Function Points Analysis: An Empirical Study of its Measurement Processes, *IEEE Transactions on Software Engineering*, Vol. 22, no 12, 1996, p. 895-909.

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

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

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

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

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

Oligny, S., P. Bourque, A. Abran. « 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.

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

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

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

Appendice: Tableau des facteurs d'équivalence

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