



LE MODELE COCOMO

BONNES PRATIQUES

CLAIRE FLEURE

Statut : V3 – Septembre 2011 - Validé





SOMMAIRE

SOMMAIRE.....	2
INTRODUCTION	4
AVERTISSEMENT	4
1. MODELE SIMPLE, COCOMO 81.....	4
1.1 UTILISATION DE COCOMO	4
1.2 TROIS TYPES DE PROJETS	5
1.2.1 Projet organique ou classique	5
1.2.2 Projet semi-détaché.....	5
1.2.3 Projet embarqué	5
1.2.4 Résumé.....	6
1.3 ESTIMATION DES CHARGES BASEE SUR LES LIGNES DE CODE	6
1.3.1 Equations de base :	6
1.3.2 Calcul des variables A, B et C pour les trois typologies de projet.....	6
2. COCOMO INTERMEDIAIRE	7
2.1 INTRODUCTION	7
2.2 ESTIMATION DES CHARGES.....	8
2.2.1 Changement des variables.....	8
2.2.2 Les paramètres d'ajustement	8
2.2.3 Le tableau des facteurs d'ajustement	9
3 COCOMO DETAILLE OU COCOMO II	9
3.1 INTRODUCTION	9
3.1.1 Niveau 1 (Early prototyping model).....	10

3.1.2	Niveaux 2 (Early design model) et 3 (Post-architecture model).....	10
3.2	PARAMETRES D'ESTIMATION	11
3.2.1	Modèles Post architecture et Early design	11
3.2.1.1	Calcul de l'effort et du délai	11
3.2.1.2	Tableaux des EMI dans le cadre du modèle Post-Architecture	12
3.2.1.3	Tableau des EMI dans le cadre du modèle Early design	25
3.2.1.4	Tableaux des SF _j	31
4	<u>CALCUL DE LA TAILLE LOGICIELLE</u>	<u>37</u>
4.1	TAILLE EN LIGNES DE CODE OU EN POINTS DE FONCTION	37
4.1.1	Lignes de Code	37
4.1.2	Les Points de Fonction	38
4.2	REUTILISATION DE COMPOSANTS	38
4.2.1	Effets non linéaires de la réutilisation	38
4.2.2	Modèle de réutilisation.....	40
4.2.2.1	Valorisation de SU: compréhension logicielle.....	41
4.2.2.2	Valorisation de AA : Evaluation et Assimilation	42
4.2.2.3	Valorisation de UNFM : méconnaissance du logiciel	42
4.2.2.4	Valorisation de AAM : Modificateur de l'Ajustement de l'Adaptation.....	43
4.2.2.5	Directives pour la quantification des logiciels adaptés.....	44
4.3	EXIGENCES : EVOLUTION ET VOLATILITE.....	46
4.4	REINGENIERIE ET CONVERSION DE CODE	46
4.5	CALCUL DE LA TAILLE DE LA MAINTENANCE LOGICIELLE	48
	<u>ANNEXE A : PASSAGE DES POINTS DE FONCTION EN LIGNES DE CODE</u>	<u>50</u>

INTRODUCTION

La méthode d'estimation COCOMO (COConstructive COSt MOdel : modèle constructif de coûts) a été introduit en 1981 par Barry Boehm. Ce modèle d'estimation des charges projet est qualifié de constructif parce qu'il permet de mieux prendre en compte la complexité logicielle et de disposer d'une meilleure appréhension de l'estimation.

Cette méthode cherche à limiter les erreurs de budget et les retards de livraison, qui sont malheureusement habituels dans l'industrie de développement logiciel.

Actualisée en 2000 (Cocomo II), la méthode repose sur l'analyse de Boehm sur les bases de données projets. Les abaques sont établis sur l'analyse de régression de ces systèmes.

Avant d'effectuer une estimation avec COCOMO, il convient de choisir le modèle à utiliser: soit le modèle simple, soit le modèle intermédiaire, soit le modèle détaillé.

AVERTISSEMENT

COCOMO II a été conçu pour être utilisable par des projets de développement en cascade ou en spirale. Pour que ces deux méthodes de développement restent raisonnablement compatibles, l'implémentation en mode cascade doit être scrupuleusement géré au niveau des risques, afin d'éviter de subir des réfections importantes, non incluses dans le modèle d'estimation.

La mise en œuvre du modèle en spirale, est utilisée en ajoutant un ensemble de jalons bien définis qui peuvent servir de points d'extrémité entre l'estimation de COCOMO II et les chiffres réels. Voir travaux de 1995 pour déterminer les jalons.

1. MODELE SIMPLE, COCOMO 81

1.1 UTILISATION DE COCOMO

La méthode Cocomo peut s'avérer très utile lors de prises de décisions financières sur l'effort de création de logiciels. Plus simplement, établir un budget et un planning pour un projet logiciel de développement ou de maintenance. La méthode fournit des abaques simples permettant de minimiser les calculs de charge. Faire la différence de coût entre l'achat d'un progiciel et le développement de celui-ci ou entre deux environnements de développement différents, cette méthode adresse la plupart des problématiques d'estimations de charge logicielle.



1.2 TROIS TYPES DE PROJETS

Cette version de la méthode offre une résolution de l'équation de charge de développement par type de projet. Cette résolution repose sur une base de 83 projets établis sur un cycle de développement en V.

1.2.1 Projet organique ou classique

Ce type de projet rassemble des projets de type « gestion », peu contraint à s'imbriquer dans un environnement existant. Les développeurs sont généralement expérimentés (technologies courantes, environnement de développement stable) et les équipes de développement sont relativement petites. La communication dans l'équipe reste de ce fait facilitée.

Cette première catégorie regroupe les projets allant jusqu'à 50 KSLOC ou plutôt 50 KDIS (*Delivered Source Instructions*), définis par Boehm pour la méthode.

Exemple de projet : compilateur

1.2.2 Projet semi-détaché

Cette catégorie regroupe les projets ayant des caractéristiques intermédiaires entre le logiciel de gestion et le logiciel embarqué. L'environnement étant plus complexe, les techniques moins habituelles, les développeurs ont un niveau d'expérience globalement moyen sur les systèmes impactés. L'équipe peut avoir une d'expérience partielle liée à certains aspects du projet.

Cette catégorie regroupe les projets allant jusqu'à la taille de 300 KDSI

Exemple de projet: la plupart des operating system complexes

1.2.3 Projet embarqué

Le Projet embarqué doit s'insérer dans son environnement. Son développement est soumis à des contraintes importantes: projets temps réel, projets sur des domaines d'avant-garde,...

Ce type de projet, logiciel et système complexes ayants des processus opérationnels abscons, doit inter-opérer fortement avec son environnement. Le coût de chaque changement est ce fait élevé.



Cette catégorie regroupe les projets de toute taille.

Exemple de projet: projets avioniques

1.2.4 Résumé

Projet	Taille	Innovation	Date limite	contraintes
Organique	Petite	légère	large	Stables
Semi- détaché	Moyenne	Moyenne	Moyenne	Moyennes
Embarqué	Grosse	Importante	Serrée	HW Complexes /interfaces client

1.3 ESTIMATION DES CHARGES BASEE SUR LES LIGNES DE CODE

1.3.1 Equations de base :

Ci-après les équations nécessaires à l'estimation des charges, vues par COCOMO:

- Effort (en Homme. Mois) = $A * (\text{taille en KSLOC})^B$
- Délai de développement = $2,5 * (\text{Effort})^C$
- Productivité = Effort/Délai

Remarque:

- L'effort (B) n'est pas proportionnel au volume de code
- Les valeurs suivent une extrapolation linéaire.
- A et B sont d'autant plus grands que la typologie est complexe

1.3.2 Calcul des variables A, B et C pour les trois typologies de projet



La résolution des variables A et B ont été comparées avec celles des modèles Watson-Felix et Putnam. Les résultats de ces trois modèles sont tout à fait comparables.

De même la résolution du facteur C a été comparée avec ceux des modèles Halstead et Watson-Felix, également compatibles.

Typologie	A	B	C
Organique	2,4	0,38	1,05
Semi- détaché	3	0,35	1,12
Incorporé	3,6	0,32	1,20

2. COCOMO INTERMEDIAIRE

2.1 INTRODUCTION

COCOMO intermédiaire ajoute à la méthode un ensemble de « Grands thèmes de coût » qui permettent d'effectuer l'évaluation subjective du produit, du matériel, du personnel et des attributs de projet. Cette prolongation considère un ensemble de quatre thèmes de coût, chacun avec un certain nombre de facteurs de coût.



2.2 ESTIMATION DES CHARGES

2.2.1 Changement des variables

Les variables A et B sont modifiées dans cette version, contrairement à C:

Typologie	A	B	C
organique	3,2	1,05	1,05
Semi détaché	3.0	1.12	1,12
Incorporé	2,8	1,20	1,20

2.2.2 Les paramètres d'ajustement

Les paramètres d'ajustement permettent de calibrer l'application afin d'obtenir une estimation de coût plus juste.

Pour cela il est nécessaire d'évaluer les 17 facteurs d'ajustement définis par la méthode.

Calculer la valeur du facteur d'ajustement EM par la table des caractéristiques projets.

EM est égale au produit des 17 facteurs.

$$EM = \prod_{17}^1$$

L'effort qui était: $PM = A * (\text{taille})^B$

Devient avec les 17 facteurs d'ajustement :

$$PM = MH \text{ nominal} * EM$$

2.2.3 Le tableau des facteurs d'ajustement

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
Attributs Produit							
RELY	Fiabilité requise						
DATA	Taille de la base de données	0,75	0,8	1	1,15	1,4	
CPLX	Complexité du produit		0,94	1	1,08	1,16	
RUSE	Réutilisation requise	0,7	0,85	1	1,15	1,3	1,65
DOCU	Documentation			1	1,11	1,3	1,66
Attributs Plateforme							
TIME	Contraintes liées au temps d'exécution			1	1,06	1,21	1,56
STOR	Contraintes liées à la taille de la mémoire		0,87	1	1,15	1,3	
PVOL	Volatilité de la plate forme		0,87	1	1,07	1,15	
Attributs Personnel							
ACAP	Compétence des analystes	1,46	1,19	1	0,86	0,71	
AEXP	Expérience du domaine d'application	1,29	1,13	1	0,91	0,8	
LTEX	Expérience dans le langage de programmation	1,42	1,17	1	0,86	0,7	
PCAP	Compétence des programmeurs	1,21	1,1	1	0,9		
PEXP	Expérience de la plate forme de développement	1,14	1,07	1	0,95		
PCON	Continuité du personnel. .	1,24	1,1	1	0,91	0,82	
Attributs Projet							
TOOL	Utilisation d'outils logiciel	1,24	1,1	1	0,91	0,83	
SITE	Développement sur des sites distants	1,23	1,08	1	1,04	1,1	
SCED	Contrainte sur le délai	1,23	1,18	1	1,04	1,1	-

3 COCOMO DETAILLE OU COCOMO II

3.1 INTRODUCTION

Le modèle détaillé incorpore toutes les caractéristiques de la version intermédiaire avec une évaluation de l'impact du coût pour chaque étape du cycle de vie du logiciel

COCOMO v2.0 résulte des travaux de raffinement de la méthode intermédiaire révisée en 1995 et 1997.

Cette version permet de calculer l'effort et la durée des projets de développement ou de maintenance. Ses buts sont:

- Assurer un coût Logiciel et un modèle d'estimation en accord avec les pratiques de développement modernes.
- Développer une base de données contenant des coûts logiciels et des outils supportant la capacité d'amélioration continue du modèle.
- fournir un cadre d'analyse quantitative, et un ensemble d'outils et de techniques, pour évaluer les effets des améliorations technologiques sur les coûts logiciels et les délais.

Le modèle COCOMO II est capable d'estimer des développements de type générateur d'applications, intégration de systèmes, ou infrastructure. Ceci comprend trois niveaux :

- 1er niveau: prendre en charge l'estimation des prototypes ou des applications ayant un effort composite
- 2nd niveau: supporter les estimations au stade précoce de conception d'un projet, quand la connaissance des coûts est minimale.
- 3ème niveau: prend en charge l'estimation post-Architecture d'un projet.

Cette version de COCOMO II met en œuvre 3 formules pour estimer l'effort, le calendrier et le coût, nécessaires pour développer un produit logiciel. Il fournit également la répartition de l'effort et le délai dans les phases de la vie du cycle logiciel du modèle de cascade ou du modèle Mbase (L'estimation des coûts logiciels est complètement décrite pour le modèle Mbase).

3.1.1 Niveau 1 (Early prototyping model)

Description de ce niveau:

- *Early prototyping model / Modèle de prototypage précoce ou composition d'applications:* Ce modèle est utilisé pour les projets fabriqués à l'aide d'outils graphiques. Il est basé sur les nouveaux 'Object Points'.

Ce niveau n'est pas traité dans ce document.

3.1.2 Niveaux 2 (Early design model) et 3 (Post-architecture model)

Description des deux niveaux:

- *Early design model / Modèle Conception précoce, ou avant projet:* Modèle de haut niveau utilisé dans des cas de développement incrémental ou d'alternative expérimentale. Le niveau de détails associé est conforme aux informations disponibles et aux besoins de précision de l'estimation.



- *Post-architecture model* / **Modèle post-architecture:**
Il s'agit d'un modèle détaillé de COCOMO II. Il est utilisé une fois que le projet est prêt à être développé, faisant suite au travail d'architecture générale. A ce stade, les informations détaillées sur le projet permettent d'obtenir une estimation de coût précis.

Ces deux modèles utilisent la même approche pour la mesure du produit (basé soit sur les lignes de code, soit sur les Points de Fonction) et pour facteurs de coûts. La réutilisation de composants peut être prise en compte dans la mesure.

3.2 PARAMETRES D'ESTIMATION

3.2.1 Modèles Post architecture et Early design

3.2.1.1 Calcul de l'effort et du délai

Ces deux modèles utilisent les mêmes formules pour estimer les charges de développement. Les formules de base sont:

Calcul de l'effort: $A = 2,94$
 $B = 0,91$

- Effort ajusté: $PM_{NS} = A * (taille)^E * EM$ NS : Nominal Schedule (délai nominal)
- $E = B + 0.01 * \sum_1^n SF_j$ $j=1$ à 5
- $EM = \prod_1^n EM_i$ $i=1$ à 16 (Post-Architecture)
 $i=1$ à 6 (Early Design)
- **Les 5 SF_j sont : PREC, FLEX, RESL, TEAM, PMAT**
- Le dernier EM_i de la liste (en rouge) n'est pas pris en compte dans l'estimation (SCED)



Calcul du délai de développement: $C = 3,67$
 $D = 0,28$

- $TDEV_{NS} = C * (PM_{NS})^F$
- $F = D + 0,2 * 0.01 * \sum_1^n SF_j \quad j=1 \text{ à } 5$
 $F = D + 0,2 * (E - B)$

3.2.1.2 Tableaux des EMI dans le cadre du modèle Post-Architecture

Le tableau descriptif permet pour chaque paramètre de se situer à partir de critères définis par la méthode.

Le tableau des valeurs permet lorsque la situation est définie, de voir les valeurs associées aux paramètres.

Tableaux descriptifs

- a. **Attribut produit : RELY Required Software Reliability/ fiabilité du logiciel requise**
 Il s'agit de la mesure de la fiabilité du logiciel, qui doit contenir les fonctions demandées en un temps défini. Si l'effet d'une défaillance logicielle est léger, l'inconvénient lié au problème est très faible. Si la vie humaine peut être impactée, le problème est très important.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
RELY	Fiabilité requise	légère	inconvenients	faible, facilement	recupérable	pertes	

- b. **Attribut produit : DATA Data Base Size / Taille de la base de données**

Ce facteur de coût tente de saisir l'effet des exigences liées aux données d'essai qui permettront de tester le programme. La côte est déterminée par le calcul de D / P , nombre d'octets dans la base de données d'essai / nombre de SLOC dans le programme. La taille de la base de données est importante à considérer de part l'effort nécessaire pour générer les données d'essai. En d'autres

termes, DATA saisie l'effort nécessaire pour assembler et entretenir les données nécessaires pour réaliser le test du programme.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
DATA	Taille de la base de données		Testing DB bytes/Pgm SLOC < 10	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$	

c. **Attribut produit : CPLX Product Complexity / Complexité produit**

La complexité est divisée en cinq domaines:

- les opérations de contrôle,
- les opérations de calcul,
- les opérations dépendant des périphériques,
- les opérations de gestion des données,
- et les opérations de gestion de l'interface utilisateur.

En utilisant le tableau suivant, sélectionnez un domaine ou une combinaison de domaines qui caractérisent le produit ou un composant du produit que vous voulez quantifier. L'évaluation de la complexité est la moyenne pondérée des valeurs des domaines sélectionnés.

C P L X	opérations de contrôle	opérations de calcul	opérations dépendant des périphériques	opérations de gestion des données	opérations de gestion de l'interface utilisateur
T r è s B a s	Code linéaire avec un peu d'opérateurs de programmation structurée, non-imbriqués: DO, CASE, IF THEN-ELSE. Composition de module simple via des appels de procédures ou de simples scripts	Evaluation d'expressions simples: Par exemple $A=B+C*(DE)$	Lecture simple, écriture avec les formats de déclarations simples	Tables simples dans la mémoire principale. Requêtes et mises à jour simples dans la base de données (COTS)	Ecrans de saisie et générateur de rapports simples
B a s	Simple emboîtement des opérateurs de programmation structurée. Généralement prédicats simples.	Evaluation d'expressions de niveau modéré: Par exemple: $D=\text{SQRT}(B^{**}2-4.*A*C)$	Aucune connaissance particulière nécessaire sur le processeur ou sur les caractéristiques des E/S de l'appareil. Les E /S sont faites au niveau GET / PUT	sous-ensembles de fichiers simples, avec aucun changement dans la structure des données, aucune édition, aucun fichier intermédiaire. Requêtes et mises à jour modérément complexes de la BDD (COTS)	Utilisation simple de constructeurs d'interfaces graphiques utilisateur (GUI)
N o m i n a l	Généralement simple emboîtement. Quelques contrôles intermodules. Des tables de décision. Rappels simples ou de passage de messages, y compris le support du traitement distribué du middleware	Utilisation de mathématiques standard et de routines statistiques. Opérations matricielles/ vectorielles de Base.	Le traitement I / O comprend la sélection du périphérique, la vérification de l'état et le traitement des erreurs	Entrée multi-fichiers et fichier unique de sortie. Changements structurels simples, éditions simples. Requêtes et mises à jour complexes de la BDD (COTS)	Utilisation simple de l'ensemble des widgets
H a u t	Opérateurs de programmation structurés, très imbriqués avec de nombreux prédicats composés. File d'attente et pile de contrôle. Homogénéité, du traitement distribué. Un seul processeur soft pour le contrôle du temps réel	Analyse numérique basique. Incorporation d'équations différentielles ordinaires. Troncation basique, Arrondis	Opérations d'I/O au niveau physique (traductions physiques d'adresses de stockage; recherches, lectures etc.) Optimisation les I/O qui se chevauchent	Triggers simples activés par le contenu de flux de données. Restructuration des données complexes.	Ensemble de widgets de développement et d'extensions. I/O voix, multimédia
T r è s h a	Codage Réentrant et Récuratif. Priorité sur la gestion des interruptions. synchronisation des tâches, des rappels complexes, processus	Analyse difficile mais numérique structurée : proche des analyses matricielles singulières,	Routines d'interruption de diagnostic, d'entretien, de masquage. Gestion de la ligne de communication.	Coordination de base de données distribuée. Triggers complexes. Recherche d'optimisation	2D/3D modérément complexes, graphiques dynamiques, multimédia

u t	hétérogènes distribués. Simple processeur de contrôle temps réel.	équations différentielles partielles. Simple parallélisation	Performance intensive des systèmes embarqués.		
E x t r ê m e m e n t h a u t	Planification de ressources multiples avec changements dynamiques de priorités. Contrôle au niveau microcode. Système distribué du contrôle temps réel	Analyse difficile et numérique non structurée: analyse très précise de bruit, des données stochastiques. Parallélisation complexe.	Système de timing dépendant du codage, opérations micro programmées. Performance critique des systèmes embarqués.	Hautement couplé, relationnel dynamique et structures d'objets. Gestion des données en langage naturel	Multimedia complexe, réalité virtuelle, interface en langage naturelle.

d. Attribut produit : RUSE : Developed for Reusability / Réutilisation du développement

Ce facteur de coût, est le montant de l'effort supplémentaire nécessaire pour construire des composants destinées à être réutilisées sur les projets actuels ou futurs. Il est lié au temps supplémentaire de conception générique du logiciel, de documentation plus élaborée, et de tests plus approfondis, afin d'assurer la réutilisation des composants dans d'autres applications.

Un développement impliquant la réutilisation impose des contraintes sur les paramètres RELY et DOCU du projet. La cotation de RELY devra être à au moins un niveau inférieur à la cote de RUSE. La cotation de DOCU devra être au moins nominale pour le niveau nominal et haute pour REUSE et au moins haute pour le niveau très haut et extrêmement haut de RUSE.

Facteurs de productivité de		Très bas	bas	Normal	haut	très haut	extrêmement haut
RU SE	Réutilisation du développement		Aucun	A travers un projet	A travers un programme	A travers une ligne de produit	A travers plusieurs lignes de produits

e. Attribut produit : DOCU Documentation Match to Life-Cycle Needs/ Besoins de documentation sur le cycle de vie

Plusieurs modèles d'estimations de logiciels ont un facteur de coût sur le niveau de documentation requise. Les coûts DOCU sont évalués en termes d'aptitude de la documentation projet à répondre aux besoins sur le cycle de vie. L'échelle de notation va de très faible (nombreux besoins du cycle de vie non couverts) à très élevé (très excessive pour les besoins du cycle de vie).

Tenter de réduire les coûts via un niveau de documentation faible (ou très faible) implique généralement des frais supplémentaires lors de la maintenance. Le fait d'être pauvre en documents augmentera l'indice de (non) compréhension du logiciel (SU).

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
DOCU	Documentation	Beaucoup de besoins du cycle de vie non couverts	Quelques besoins du cycle de vie non couverts	Besoins du cycle de vie couverts	Documentation excessive / besoins	Documentation très excessive / besoins	

f. Attribut Plateforme : TIME Execution Time Constraint/ contrainte du temps d'exécution

Il s'agit d'une mesure de la contrainte du temps d'exécution infligée à un système logiciel. La notation est liée au pourcentage de temps d'exécution disponible pour un système, ou un sous-système.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
TIME	Contraintes sur le temps d'exécution			≤ 50% du temps d'exécution disponible est utilisé	70% du temps d'exécution disponible est utilisé	85% du temps d'exécution disponible est utilisé	95% du temps d'exécution disponible est utilisé

g. Attribut Plateforme : STOR Main Storage Constraint / contrainte de la mémoire principale

Cette mesure représente le degré de contrainte d'un système logiciel ou un sous- système sur la mémoire principale. Compte tenu de l'augmentation importante du temps d'exécution des processeurs, disponibles sur le marché et de mémoire principale, on peut se demander si ces contraintes sont toujours d'actualité. Cependant, de nombreuses applications continuent de s'étendre et de consommer toutes les ressources disponibles (particulièrement avec la croissance des COTS lourds), rendent ces facteurs de coûts toujours d'actualité.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
STOR	Contraintes sur la mémoire centrale			Utilisation de ≤ 50% de la mémoire disponible	Utilisation de 70% de la mémoire disponible	Utilisation de 85% de la mémoire disponible	Utilisation de 95% de la mémoire disponible

h. Attribut Plateforme : PVOL Platform Volatility / Volatilité de la plateforme

Le mot plateforme est utilisé ici pour désigner un ensemble de systèmes et logiciels (OS, SGBD, etc.), permettant à une application logicielle de s'acquitter de ses tâches. Si le logiciel à développer est une application système, alors dans ce cas la plate-forme sera le matériel informatique. Pour le développement d'un système de gestion de base de données, la plateforme est système, munie d'un Operating System et de bases de données. Cette plateforme inclue un certain nombre de compilateurs, d'assembleurs pour supporter le développement.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
PVOL	Volatilité de la plateforme			Changement entre: Max :Ts les 12 mois Min : Ts les mois	Changement entre: Max :Ts les 6 mois Min : Ts les 2 sem.	Changement entre: Max :Ts les 2 mois Min : Ts les 1 sem.	Changement entre : Max :Ts les 2 sem. Min : Ts les 2 j



i. Attribut Personnel : Analyst Capability (ACAP)/ Compétence des analyses

Les analystes sont des personnes qui travaillent sur les besoins, de conception de haut niveau à la conception détaillée.

Les attributs majeurs qui devraient être pris en compte dans ce classement sont l'analyse et la capacité de conception, l'efficacité et la rigueur, et la capacité à communiquer et à coopérer.

La note ne doit pas prendre en compte le niveau d'expérience de l'analyste, déjà considérée avec les paramètres APEX, LTEX, et PLEX. Les équipes d'analystes qui ont une cotation de 15% sont jugés très faibles et ceux qui ont 90% sont très performants.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
ACAP	Compétence des Analystes	15%	35%	55%	75%	90%	

j. Attribut Personnel : Programmer Capability (PCAP)/ Compétence des programmeurs

Les tendances actuelles continuent de souligner l'importance de disposer des analystes très compétents.

Toutefois, le rôle croissant de progiciels complexes (COTS), et la demande de productivité effectuent une contrainte de plus en plus importante sur la compétence des programmeurs. L'évaluation doit être fondée sur la compétence de l'équipe de programmeurs plutôt que sur les compétences individuelles. Les principaux facteurs qui devraient être pris en compte dans la notation sont la compétence, l'efficacité et la rigueur et la capacité à communiquer et à coopérer.

La note ne doit pas compte du niveau d'expérience du programmeur, déjà considérée avec les paramètres APEX, LTEX, et PLEX.

Les équipes de programmeurs qui ont une cotation de 15% sont jugés très faibles et ceux qui ont 90% sont très performants.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
PCAP	Compétence des Programmeurs	15%	35%	55%	75%	90%	

k. Attribut Personnel : Personnel Continuity (PCON)/ Rotation du personnel (turnover)

L'échelle de notation est exprimée en pourcentage de rotation du personnel sur une durée de un an: à 3%, très bonne stabilité, à 48%, rotation importante.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
PCON	Rotation du personnel	48%	24%	12%	6%	3%	

l. Attribut Personnel : Applications Experience (APEX)/ Expérience du domaine applicatif

L'échelle de cette notation (anciennement appelée AEXP) est dépendant de l'expérience de l'équipe de développement sur les systèmes et sous-systèmes logiciels.

Les notes sont définies en fonction du niveau d'expérience de l'équipe projet sur le type d'application. Une note très faible pour une expérience de moins de 2 mois et très haute pour une de 6 ans ou plus.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
APEX	Expérience du domaine applicatif	≤ 2 mois	6 mois	1 an	3 ans	6 ans	

m. Attribut Personnel : Platform Experience (PLEX)/ Expérience sur la plateforme de développement

PLEX (anciennement étiquetés PEXP), dans le modèle post-Architecture élargit l'influence de la plateforme, en reconnaissant l'importance de maîtriser l'utilisation de celles-ci. En effet, les plateformes sont de plus en plus puissantes, et incorporent des interfaces utilisateur graphiques, des bases de données, des réseaux, et les capacités du middleware distribuées.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
PLEX	Expérience sur la plateforme	≤ 2 mois	6 mois	1 an	3 ans	6 ans	

n. Attribut Personnel : Language and Tool Experience (LTEX)/ Expérience sur le langage et les outils

Cette échelle de notation mesure la connaissance et l'expérience, de l'équipe projet, sur le langage de programmation et les outils logiciels. Le développement logiciel s'appuie sur l'utilisation d'outils nécessaires à : la gestion des exigences, la conception et l'analyse, la configuration, la gestion projet, l'extraction de documents, la gestion des bibliothèques, la vérification de la cohérence, la planification, les tests, etc.

Une faible note est donnée pour une expérience de moins de 2 mois. Une note très élevée est accordée pour une expérience de 6 ans ou plus.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
LTEX	Expérience Langage et outils	≤ 2 mois	6 mois	1 an	3 ans	6 ans	

o. Attribut Projet : Use of Software Tools (TOOL)/ Utilisation d’outils logiciels

Les outils logiciels, utilisés pour calibrer la version 1981 de COCOMO, ont été nettement améliorées depuis les années 1970». La cote de l'outil varie du simple « codage et modification de celui-ci » (coté très faible), à des outils intégrés permettant de gérer le cycle de vie projet (coté très élevé).

Une note nominale dans le paramètre OUTIL en COCOMO 81 est équivalente à une cote très faible dans celui de COCOMO II.

Une extension prochaine de COCOMO serait de procéder à l'élaboration de l'échelle de notation du paramètre OUTIL et de briser les effets de capacités, maturité et intégration.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
T O O L	Utilisation outils logiciels	edit, code, debug	simple, frontend, backend CASE, petite intégration	Outils basiques du cycle de vie. Intégration modérée	Outils du cycle de vie importants et matures Intégration modérée	Outils du cycle de vie importants, matures et proactif. Intégration bien intégrée avec des procédés, des méthodes, de la réutilisation	

p. Attribut Projet : Multisite Development (SITE)/ Développement multi sites

Le facteur de coût SITE a été ajouté dans COCOMO II, ces effets sur le développement étant importants.

Déterminer sa note nécessite une évaluation et un jugement associé basés sur une moyenne de deux facteurs :

La collocalisation du site (allant d’une collocalisation totalement à une distribution internationale) et de la communication de soutien (du courrier de surface et des accès téléphoniques au multimédia interactif).

Par exemple, si une équipe est complètement en collocalisation, il n'a pas besoin de multimédia

interactif pour obtenir une note très haute. Une messagerie électronique sera généralement suffisante.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
S I T E	collocalisation	International	Villes différentes et multi compagnies	Villes différentes ou multi compagnies	Même ville ou même agglomération	Même immeuble ou zone	collocalisation
S I T E	communication	Quelques téléphones, mail	téléphone, FAX individuels	Courriel à bande étroite	Communication électronique bande large	Com. électronique bande large Vidéo conférences occasionnelles	Multi média interactif

q. Attribut Projet : Required Development Schedule (SCED)/ Délai de développement requis

Cette note mesure la contrainte liée au calendrier imposé à l'équipe du projet de développement logiciel. Les notations sont définies en termes de pourcentage de décélération ou d'accélération du temps par rapport au calendrier nominal du projet, représentant une quantité donnée d'effort. Calendriers accélérés ont tendance à produire:

- plus d'efforts dans les premières phases pour éliminer les risques et affiner l'architecture
- plus d'efforts dans les phases ultérieures pour accomplir davantage de tests et de documents en parallèle.

Dans le tableau suivant, la compression du temps de 75% est jugée très faible. Un étirement du délai 160% est classé très haut. Etirer ou compresser revient à ajouter ou diminuer l'effort. Ceci est épargné pour une équipe plus petite, pour laquelle le délai est généralement équilibré par le besoin de mener les fonctions administratives projet sur une plus longue période de temps. La nature de cet équilibre est en cours d'étude.



SCED est le seul facteur coût utilisé pour décrire l'effet de la compression/expansion du délai pour l'ensemble du projet. Les facteurs d'échelle sont également utilisés pour décrire l'ensemble du projet.

Tous les autres facteurs de coût sont utilisés pour décrire chaque module pour un projet en ayant plusieurs. Utilisation du COCOMO II Post-architecture modèle pour l'estimation des modules multiples est expliquée plus loin.

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
SCED	Délai de développement requis	75% / normal	85% / normal	100% / normal	130% / normal	160% / normal	

Tableau des valeurs

Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
Attributs Produit							
RELY	Fiabilité requise	0,82	0,92	1,00	1,10	1,26	
DATA	Taille de la base de données		0,90	1,00	1,14	1,28	
CPLX	Complexité du produit	0,73	0,87	1,00	1,17	1,34	1,74
RUSE	Réutilisation requise		0,95	1,00	1,07	1,15	1,24
DOCU	Documentation	0,81	0,91	1,00	1,11	1,23	
Attributs Plateforme							
TIME	Contraintes liées au temps d'exécution			1,00	1,11	1,29	1,63
STOR	Contraintes liées à la taille de la mémoire			1,00	1,05	1,17	1,46
PVOL	Volatilité de la plate forme		0,87	1,00	1,15	1,30	
Attributs Personnel							
ACAP	Compétence des analystes	1,42	1,19	1,00	0,85	0,71	
PCAP	Compétence des programmeurs	1,34	1,15	1,00	0,88	0,76	
PCON	Continuité du personnel. .	1,29	1,12	1,00	0,90	0,81	
APEX	Expérience du domaine d'application	1,22	1,10	1,00	0,88	0,81	
PEXP	Expérience de la plate forme de développement	1,19	1,09	1,00	0,91	0,85	
LTEX	Expérience dans le langage de programmation	1,20	1,09	1,00	0,91	0,84	
Attributs Projet							
TOOL	Utilisation d'outils logiciels	1,17	1,09	1,00	0,90	0,78	
SITE	Développement sur des	1,22	1,09	1,00	0,93	0,86	0,80

	sites distants						
SCED	Contrainte sur le délai	1,43	1,14	1,00	1,00	1,00	-

3.2.1.3 Tableau des EMI dans le cadre du modèle Early design

Ce modèle est utilisé dans les premières phases d'un projet logiciel alors que très peu de choses sont définies que ce soit au sujet de la taille du produit devant être développé, de la nature de la plateforme cible, du type de personnel impliqué, ou des spécificités détaillées des processus à utiliser.

Ce modèle pourrait être utilisé dans le développement de générateur d'application, l'intégration de systèmes, ou dans les secteurs de développement de l'infrastructure.

Le modèle Early Design utilise pour la valorisation de la taille du logiciel à développer, la mesure des Points de Fonction ou des lignes de codes (KSLOC).

L'application de facteurs d'échelle exponentiels est le même pour les modèles Early Design et Post-Architecture. Dans le modèle Early Design le jeu de facteurs de coût est réduit. Les paramètres d'Early Design sont obtenus en combinant ceux du modèle Post-Architecture. Chaque fois que l'évaluation d'un facteur de coût est à mi chemin entre deux niveaux, toujours arrondir à la cote nominale.

Par exemple: Si une évaluation conclut à une valeur à mi chemin entre très faible et faible, sélectionnez Faible.

L'équation de l'effort est identique:

$$PM_{NS} = A * (taille)^E * EM$$

$$EM = \prod_1^n EM_i$$

Sauf que le nombre de multiplicateurs d'effort est réduit à 7 (n = 7).



Définition des paramètres d'Early Design et en fonction de Post-Architecture

Early Design	Post-Architecture
PERS	ACAP, PCAP, PCON
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

Approche globale:

L'approche suivante est d'utiliser les échelles de notation et les facteurs de coûts de Post-Architecture et les translater sur leurs homologues d'Early Design. Cela implique l'utilisation et la combinaison des équivalents numériques des niveaux de classification. Précisément, une notation de facteurs coût en Post-Architecture correspondant à une cote numérique:

1 pour très bas, 2 Bas, 3 Nominal, 4 Haute, 5 très haute et 6 extrêmement haut.

Pour les facteurs de coûts combinés Early Design, les valeurs numériques des pilotes constituants de post-Architecture sont additionnés, et le résultat final est alloué à l'échelle élargie d'Early Design, allant de Extrêmement bas à Très élevé. Les échelles de notation du modèle Early Design ont toujours un nominal total égal à la somme des cotes nominales des éléments de sa contribution. Un exemple est donné ci-dessous pour l'inducteur de coût PERS

Tableau descriptif

Le tableau descriptif permet pour chaque paramètre de se situer à partir de critères définis par la méthode.

a) Personnel Capability (PERS) and Mapping Example / Capacité du personnel (PERS) et l'exemple de cartographie

Le paramètre PERS dans Early Design combine la capacité des analystes (ACAP), celle des Programmeurs (PPCE), et la continuité du personnel (PCON) de post-Architecture. Chacun d'eux a une échelle de notation de Très Faible (= 1) à Très élevé (= 5). L'addition de leurs cotes numériques produit des valeurs allant de 3 à 15.



Les multiplicateurs d'effort associés sont dérivés de celles d'ACAP, de PPCE, et de PCON en effectuant la moyenne des produits de chaque combinaison de multiplicateurs d'effort associés. Les multiplicateurs d'effort pour PERS, comme les autres pilotes du modèle Early Design, sont dérivées de celles du modèle post-Architecture (dans ce cas PACS, PPCE, PCON), et la note est obtenue en faisant la moyenne.

PERS = Extrêmement Haute impliquerait les combinaisons: PACS, PPCE, et PCON tous très élevés, ou seulement l'un des paramètres Haut et les deux autres très Hauts.

PERS	Extrêmement Bas	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Somme de : ACAP, PCAP, PCON	3,4	5,6	7,8	9	10,11	12,13	14, 15
Combinaison de ACAP et PCAP	20%	35%	45%	55%	65%	75%	85%
Turnover annuel	45%	30%	20%	12%	9%	6%	4%

La puissance nominale (9) de PERS correspond à la somme (3 + 3 + 3) de la cote nominale pour le PACS, le PPCE, et PCON.

Notez cependant que la valeur nominale de 9 de PERS peut résulter d'un certain nombre d'autres combinaisons, par exemple 1 + 3 + 5 = 9 pour le PACS = très faible, PPCE = nominale, et PCON = Très élevé.

Les échelles de notation et des multiplicateurs d'effort pour PCAP et des autres inducteurs de coûts d'Early Design sont fortement liées avec leurs homologues de Post-Architecture.

Par exemple:

PERS ayant une notation extrêmement faible (20% de l'ACAP et PPCE; turnover de 45%) présente des moyennes d'ACAP, de PPCE, et de PCON de 3 ou 4.

Le maintien de ces relations entre Early Design et Post-Architecture garantit la cohérence des coûts



d'estimations. Il permet également une échelle de notation aux facteurs individuels de coûts de post-Architecture.

b) Product Reliability and Complexity (RCPX)/ Fiabilité des produits et la complexité

Ce pilote de coûts d'Early Design combine quatre facteurs de coûts de Post-Architecture : la fiabilité des logiciels (FIER), la taille de la base de données (DATA), la complexité du produit (CPLX), et le besoin de documentation sur le cycle de vie (DOCU).

Contrairement au composant PERS, le composant RCPX est composé avec des échelles de notation ayant des largeurs différentes. RELY et DOCU ont une échelle de notation de Très bas à Très-Haut; DATA de Bas à Très-Haut, et CPLX de Très bas à Extrêmement Haut. La somme numérique de leurs notations varie ainsi de 5 (TB, B, TB, TB) à 21 (EH, TH, EH, TH).

Comme avec PERS, les facteurs de coût RELY, DATA CPLX et DOCU de Post-Architecture, et des échelles de notation permettent de fournir une sauvegarde détaillée pour interpréter la notation RCPX d'Early Design.

RCPX	Extrêmement Bas	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Somme de RELY, DATA, CPLX, DOCU	5,6	7,8	9-11	12	13-15	16-18	19-21
Accent mis sur la fiabilité, la documentation	Très peu	peu	quelques	basique	forte	Très forte	Extrêmement forte
Complexité produit	Très simple	simple	quelques	moyenne	complexe	Très complexe	Extrêmement complexe
Taille de la base de données	petite	petite	petite	modérée	Grosse	Très grosse	Très grosse

c) Developed for Reusability (RUSE) / développé pour être réutilisé

Le facteur de coût d’Early Design est le même que celui de Post-Architecture.

d) Platform Difficulty (PDIF) / Difficulté liée à la plateforme

Le facteur de coût d’Early Design combine trois facteurs de coût de Post-Architecture: La contrainte sur le temps d’exécution (TIME), la contrainte de stockage principal (STOR), et la volatilité de la plateforme (PVOL). TEMPS et STOR ont une échelle de notation du Nominal à Très haut; celle de PVOL de Bas à Très Haut.

La somme numérique de leurs notations varie ainsi de 8 (N, N, B) à 17 (EH, EH, TH).

PDIF	Extrêmement Bas	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Somme de TIME, STOR, PVOL			8	9	10-12	13-15	16,17
Temps et contrainte de stockage			≤50%	≤50%	65%	80%	90%
Volatilité de la plateforme			Très stable	Stable	Quelques volatilités	volatile	Hautement volatile

e) Personnel Experience (PREX) / Expérience du personnel

Ce facteur de coût d’Early Design combine trois facteurs de coût de Post-Architecture : L’expérience sur l’application (APEX), L’expérience sur le langage et les outils (LTEX), et L’expérience sur la plateforme(PLEX). Chacune ont une échelle de notation allant de Très bas à Très haut. La somme numérique de leurs notations varie ainsi de 3 à 15.

PREX	Extrêmement Bas	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
------	-----------------	----------	-----	---------	-------	------------	-------------------

Somme de APEX, LPEX, PLEX	3,4	5,6	7,8	9	10,11	12,13	14, 15
Expériences sur: les Applications, la Plateforme, le Langage et les outils	≤ 3 mois	5 mois	9 mois	1 an	2 ans	4 ans	6 ans

f) Facilities (FCIL) /Facilités

Ce facteur de coût d'Early Design combine deux facteurs de coût de Post-Architecture: Utilisation d'outils logiciels (TOOL) et développement multi-sites (SITE). TOOL a une échelle de notation allant de Très bas à Très Haut, Site de Très Bas à Extrêmement Haut.

FCIL	Extrêmement Bas	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Somme de TOOL et SITE	2	3	4,5	6	7,8	9,10	11
Support d'outils	Minimum	Quelques	Petite collection	Outils de base du cycle de vie	Bonne, modérément intégrés	Forte modérément intégrés	Forte Bien intégrés
Conditions du multi-site	faible support du développement multi sites complexes	Quelques support sur développ. Maint /Support complexes	Quelques supports sur dev. M/S légèrement complexe	Support de base sur Dév. M/S légèrement complexe	Fort support sur Dév. M/S légèrement complexe	Fort support sur développ. M/S simple	Très fort support sur développ. M/S collocalisé ou simple

g) Required Development Schedule (SCED) / Délai de développement exigé

Le facteur de coût d'Early Design est le même que celui de Post-Architecture.

Tableau des valeurs

Le tableau des valeurs permet lorsque la situation est définie, de voir les valeurs associées aux paramètres.

		Extrêmement bas	Très bas	bas	Normal	haut	très haut	extrêmement haut
Attributs Produit								
PCPX	Fiabilité et complexité	0,49	0,60	0,83	1,00	1,33	1,91	2,72
RUSE	Réutilisation requise			0,95	1,00	1,07	1,15	1,24
Attributs Plateforme								
PDIF	Difficulté liée à la plateforme			1,00	1,00	1,00		
Attributs Personnel								
PERS	compétence de l'équipe	2,12	1,62	1,26	1,00	0,83	0,63	0,50
PREX	Expérience de l'équipe	1,59	1,33	1,12	1,00	0,87	0,74	0,62
Attributs Projet								
FCIL	Facilités	1,43	1,30	1,10	1,00	0,87	0,73	0,62
SCED	Contrainte sur le délai		1,43	1,14	1,00	1,00	1,00	

3.2.1.4 Tableaux des SF_j

Le tableau ci-après permet de valoriser les cinq facteurs d'échelle SF_j.

La note finale est la moyenne pondérée subjective des caractéristiques énumérées.

Tableaux descriptifs

a. Précédence (PREC)

Si un produit est semblable à plusieurs projets précédemment développés, dans ce cas la précédence est haute



PREC /SF1	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Compréhension de l'organisation sur les objectifs produit	Générale		Bonne			Approfondie
Expérience de travail avec systèmes logiciels associés	Modérée		Considérable			Etendue
Développement simultané de Concurrent de nouveaux composants système et de procédures opérationnelles	Etendu		Modéré			Certain
GENERAL	sans précédent	Souvent sans précédent	Peu sans précédent	En général familier	Largement familier	Complètement familier

FLEX /SF2	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Besoin d'assurer la conformité du logiciel avec les exigences préétablies	complètement		considérable			De base
Besoin d'assurer la conformité du logiciel avec les spécifications des interfaces externes	complètement		considérable			De base
Combinaison de rigidités ci-dessus avec prime sur l'achèvement rapide	Haut		Moyen			Bas



GENERAL	rigoureux	Relâche occasionnelle	Un peu de relâche	En général conforme	Quelques conformités	Buts généraux

b. Flexibilité de développement (FLEX)

Les facteurs d'échelle PREC et FLEX sont en grande partie intrinsèques au projet et incontrôlables. Les trois facteurs d'échelle suivants proposent une gestion contrôlable de projet. Ils permettent d'effectuer des économies d'échelle en réduisant les sources de turbulence du projet, l'entropie, et le redéveloppement.

c. Architecture /risque de résolution (RESL)

Ce facteur regroupe deux des facteurs d'échelle, " Design Thoroughness by Product Design Review (PDR) " et " Risk Elimination by PDR "

Sigles utilisés :

RDP : Product Design Review : revue du design du produit

LCA : Life Cycle Architecture : Architecture du cycle de vie

RESL/SF3	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Le plan de gestion des risques identifie tous les éléments de risque critique, établit des jalons pour les résoudre par la RDP ou	N'existe pas	Un peu	Quelque	En général	La plupart du temps	complètement



LCA						
Le délai, le budget, et les étapes internes à travers RDP ou LCA, sont compatibles avec Plan de gestion des risques	N'existe pas	Un peu	Quelque	En général	La plupart du temps	complètement
Pourcentage du délai de développement consacré à l'architecture compte tenu des objectifs généraux du produit	5	10	17	25	33	40
Pourcentage requis d'architectes logiciels expérimentés disponibles pour le projet	20	40	60	80	100	120
Outils d'aide disponibles pour résoudre les éléments de risque, développer et vérifier les spécifications d'architecture	Aucun	Peu	Quelque	Bon	Fort	Complet
Niveau d'incertitude des composants clés de l'architecture: la mission, l'interface utilisateur, les COTS, le matériel, la technologie, la performance	Extrême	Significatif	Considérable	Quelques	Un peu	Très peu
Nombre et criticité des points de risque	> 10 Points critiques	5-10 Points critiques	2-4 Points critiques	1 Point critique	> 5 Non-Points critiques	< 5 Non-Points critiques
GENERAL	Un petit peu 20%	Un peu 40%	Souvent 60%	Général 75%	La plupart du temps 90%	Complètement 100%

d. Cohésion de l'équipe (TEAM)

Le montant du facteur d'échelle Cohésion d'équipe représente l'indiscipline et la confusion du projet dues aux difficultés de synchroniser les parties prenantes: utilisateurs, clients, développeurs,



support et autres. Ces difficultés peuvent résulter de différences dans les objectifs, la culture, des difficultés à concilier les objectifs et le manque d'expérience de l'équipe,...

TEAM /SF4	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Cohérence des objectifs des intervenants et des cultures	Un peu	Quelque	Basique	Considérable	Forte	Complète
Capacité volonté, des intervenants à tenir compte des objectifs des autres parties prenantes	Un peu	Quelque	Basique	Considérable	Forte	Complète
Expérience des intervenants dans le travail en équipe	Aucune	Un peu	Quelque	Basique	Considérable	Vaste
Esprit d'équipe des intervenants pour créer une vision partagée et des engagements	Aucune	Un peu	Quelque	Basique	Considérable	Vaste
GENERAL	Interactions très difficiles	Interactions un peu difficiles	Coopération de base	Assez bonne coopération	Bonne coopération	Interactions très facile

e. Maturité de processus (PMAT)

PROC/SF5	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
Estimated Process Maturity Level (EPML)	0	1	2	3	4	5
GENERAL	CMMI 1 Faible	CMMI 1 Fort	CMMI 2	CMMI 3	CMMI 4	CMMI 5

Tableau des valeurs

Facteurs d'échelles	Très bas	Bas	Nominal	Grand	Très grand	Extrêmement grand
PREC /SF1	6,20	4,96	3,72	2,48	1,24	0
FLEX /SF2	5,07	4,05	3,04	2,03	1,01	0
RESL3 /SF3	(20%) 7,07	(40%) 5,65	(60%) 4,24	(75%) 2,83	(90%) 1,41	(100%) 0
TEAM/SF4	5,48	4,38	3,29	2,19	1,1	0
PMAT/SF5	7,80	6,24	4,68	3,12	1,56	0

4 CALCUL DE LA TAILLE LOGICIELLE

Une bonne estimation de la taille logicielle est très importante pour une bonne estimation des charges. Cependant, déterminer la taille peut être difficile. Les projets sont généralement composés d'une part de nouveaux développements, et d'autre part de code réutilisé provenant de divers sources - avec ou sans modifications - et de code automatiquement traduit d'un langage à l'autre.

COCOMO II utilise uniquement les éléments de la taille qui influencent l'effort de développement : le nouveau code, le code copié et le code modifié.

Une technique est utilisée pour rendre le nouveau code et les codes réutilisés équivalents, afin de pouvoir contribuer à l'estimation de la taille globale. Dans base de référence (Baseline) de COCOMO, la taille est un comptage de nouvelles lignes de code. Le comptage du code copié, puis modifié doit être ajusté afin d'obtenir un équivalent en nouvelles lignes de code. L'ajustement prend en compte la quantité de conception, codage et de tests qui ont été changés. Il considère également la compréhensibilité et la connaissance ou non du code.

Pour le code automatiquement traduit, un taux de productivité de traduction est utilisé pour déterminer l'effort nécessaire correspondant à la quantité de code à traduire.

4.1 TAILLE EN LIGNES DE CODE OU EN POINTS DE FONCTION

4.1.1 Lignes de Code

Il y a différentes sources permettant de calculer le nombre de Lignes de Code. La meilleure source est basée sur des données historiques. Par exemple, en début de projet, les spécifications peuvent être mesurées en Points de Fonction, en composants, ou en une autre unité qui permet d'estimer les lignes de code. Dans le cas des Points de Fonction, voir tableau de conversion en lignes de code en Annexe.

Faute de données historiques, les dires d'experts peuvent être utilisés pour estimer la taille en trois points: taille minimum, taille probable, taille maximum.

La taille du code est exprimée en milliers de lignes de code source de (KSLOC). Une ligne de source de code exclue généralement tout logiciel nécessaire au support, non livrée, tel que les drivers de tests.

Toutefois, si ces derniers sont développés avec le même soin que les logiciels livrés, avec leurs propres revues, leurs plans de test et documentation, alors ils devraient être comptés [Boehm,

1981, p. 58-59]. L'objectif est de mesurer la quantité de travail intellectuel nécessaire à l'élaboration de logiciels.

Le SEI (Software Engineering Institute) a mis à disposition une liste permettant de définir la « déclaration de source logique », utilisée comme définition de la ligne de mesure de code. Dans COCOMO II, cette définition a été choisie comme standard pour la ligne de code.

4.1.2 Les Points de Fonction

La méthode des Points de Fonction est basée sur la quantité de fonctionnalités et de données d'un projet logiciel et éventuellement un ensemble de facteurs d'ajustement. Les Points de Fonction sont des mesures utiles de la taille logicielle car ils sont basés sur l'information qui est disponible tôt dans le cycle de vie projet.

Une fois les Points de Fonction calculés, ils sont converti en Lignes de code, en fonction du langage utilisé, car COCOMO n'est utilisable qu'avec des Lignes de Code. Voir Annexe

La définition des Points de Fonction est diffusée par l'IFPUG : <http://www.ifpug.org/>

4.2 REUTILISATION DE COMPOSANTS

L'estimation de charge logicielle vue jusqu'à présent concerne le développement d'un nouveau produit. Le code qui est repris et intégré dans le nouveau développement contribue également à la charge. Le code préexistant, qui est traité comme une boîte noire, est appelé code réutilisé. Le code préexistant qui est traité comme une boîte blanche et est modifié pour être réutilisé, est appelé code adapté. La taille effective du code réutilisé et adapté est ajustée pour obtenir son équivalence en nouveau code. Le code ajusté est appelé lignes de code source équivalente (ESLOC). L'ajustement est basé sur l'effort supplémentaire qu'il faut pour modifier le code et l'inclure dans le produit.

Le modèle traite le dimensionnement de la réutilisation (à partir des points de fonction ou des lignes de source de Code), de la même façon dans le modèle Early Design et le modèle Post-Architecture.

4.2.1 Effets non linéaires de la réutilisation

L'analyse des coûts de réutilisation [Selby 1988], pour près de trois mille modules réutilisés de la NASA Software Engineering Laboratory, indique que la fonction du coût de réutilisation n'est pas

linéaire. Cette non-linéarité est significative et impliquée par les charges de modification du code réutilisé et le coût résultant de la réutilisation. L'effort requis pour la réutilisation de code n'est jamais nul. Il y a généralement un coût d'environ 5% pour évaluer, sélectionner et assimiler les composants réutilisables.

D'autre part le code réutilisable est très souvent modifié, même lorsque celui-ci devait être une boîte noire. Or les petites modifications dans le produit réutilisé génèrent des coûts importants et disproportionnés. C'est principalement à cause de deux facteurs: le coût de la compréhension du logiciel devant être modifié, et le coût relatif de la vérification des interfaces du module.

[Parikh-Zvegintzov 1983] contient des données montrant que 47% de l'effort de la maintenance logicielle consiste à comprendre le logiciel devant être modifié. Ainsi, dès que l'on va à partir d'une boîte noire réutilisée la transformer en boîte blanche, on rencontre cette sanction de compréhension du logiciel.

En outre, [Gerlich-Denskat 1994] montre l'équation suivante :

k: quantité de code modifiée k contenu dans les modules réutilisés du logiciel **m**,

N : le nombre d'interfaces impactées dont les vérifications sont nécessaires, N.

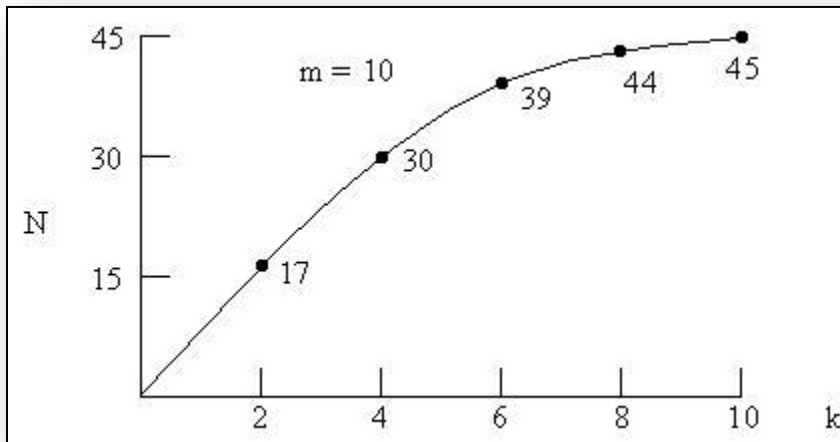
$$N = k \times (m - k) + k \times \left[\frac{k-1}{2} \right]$$

Exemple

m = 10 modules.

La modification impacte 20% des modules (2 sur 10)

La revalidation est de 38% (17 sur 45) sur les interfaces.



La forme de cette courbe est similaire pour d'autres valeurs de m. Cela indique qu'il existe des effets non linéaires impliqués dans la vérification du module d'interface et qui se produisent lors de la conception, du codage, de l'intégration et de test des logiciels modifiés.

4.2.2 Modèle de réutilisation

Le traitement de COCOMO de la réutilisation du logiciel utilise un modèle d'estimation non linéaire, (Voir équations suivantes). Cela implique l'estimation de la quantité de logiciels à être adapté et les trois facteurs de degrés de modification : le pourcentage de modification de la conception (DM), le pourcentage de code modifié (CM), et le pourcentage de l'effort d'intégration requis pour intégrer le logiciel adapté ou réutilisé (GI).

$$\text{KSLOC Equivalents} = \text{KSLOC Adaptés} \times \left(1 - \left[\frac{\text{AT}}{100}\right]\right) \times \text{AAM}$$

$$\text{AAM} = \frac{[\text{AA} + \text{AAF} \times (1 + (0,02 \times \text{SU} \times \text{UNFM}))]}{100} \quad \text{Pour AAF} \leq 50$$

$$\text{AAM} = \frac{[\text{AA} + \text{AAF} + (\text{SU} \times \text{UNFM})]}{100} \quad \text{Pour AAF} > 50$$

$$AAF = (0,4 \times DM) + (0,3 \times CM) + (0,3 \times IM)$$

4.2.2.1 Valorisation de SU: compréhension logicielle

L'incrément de compréhension logicielle (**SU**) est obtenu à partir du tableau suivant.

SU est exprimé en pourcentage. Si le logiciel est classé très haut sur la structure, la clarté de l'application, l'auto-description, et la compréhension de l'interface de contrôle est pénalisé à 10%. Si le logiciel est classé très bas sur ces facteurs, la pénalité est de 50%.

SU est déterminé en prenant la moyenne subjective des trois catégories.

	Très bas	Bas	Nominal	Haut	Très haut
Structure	Cohésion très faible, Fort couplage code spaghetti	Cohésion Modérément faible. Fort couplage	Raisonnement bien structuré; certaines zones de faiblesse	Haute cohésion ; couplage faible	Forte modularité ; informations cachés dans les données / contrôle de structures
Clarté de l'application	aucune correspondance entre le programme et les vues (écrans) de l'application	Quelques corrélations entre le programme et l'application	Des corrélations entre le programme et l'application	Bonne corrélations entre le programme et l'application	Correspondance entre le programme et les vues (écrans,...) de l'application
Auto-description	Code obscur, documentation absente, obscure ou obsolète	Quelques commentaires dans le code et des en-têtes; certaines documentations utiles	Des commentaires dans le code et des en-têtes; documentation	Bons commentaires dans le code et en-têtes; documentations utiles ; certaines zones de faiblesse	code Auto-descriptif; documentation mise à jour, bien organisé, avec des copies d'écrans en justification.
Incrément SU pour les ESLOC	50%	40%	30%	20%	10%



4.2.2.2 Valorisation de AA : Evaluation et Assimilation

L'incrément de réutilisation non linéaire est traité en degré d'évaluation et Assimilation (**AA** : Assessment and Assimilation) nécessaires pour déterminer si un module logiciel réutilisé est adapté à la demande, et pour intégrer sa description dans la description générale du produit. Le tableau suivant fournit l'échelle de notation et les valeurs correspondantes pour l'incrément Evaluation et Assimilation. AA est un pourcentage.

Incrément AA	Niveau d'effort lié à AA
0	Aucun
2	Recherche dans le module de base et la documentation
4	Certains modules de test et d'évaluation (T&E), de la documentation
6	Important module T&E, documentation
8	Module T&TE complet, documentation

4.2.2.3 Valorisation de UNFM : méconnaissance du logiciel

Le montant de l'effort requis pour modifier le logiciel existant est une fonction non seulement liée au montant de la modification (AAF) et à l'intelligibilité du logiciel existant (SU), mais aussi à la relative méconnaissance du logiciel (**UNFM** : unfamiliarity). Le facteur UNFM est appliqué en le multipliant à l'incrément de compréhension logicielle (SU). Si le programmeur travaille avec le logiciel tous les jours, le multiplicateur d'UNFM sera 0,0. Si le programmeur n'a jamais vu le logiciel avant, le multiplicateur de 1,0 valorisera pleinement l'incrément de compréhension du logiciel.

Incrément UNFM	Niveau de non-familiarité
0,0	Familier
0,2	Partiellement familier
0,4	peu familier
0,6	Un peu non-familier
0,8	Plutôt non familier
1	Complètement non familier



4.2.2.4 Valorisation de AAM : Modificateur de l'Ajustement de l'Adaptation

L'équation de réutilisation de modules est utilisé pour déterminer une équivalence au travail à effectué, quantifié en nouvelles lignes de code source. Le calcul d'équivalence SLOC est basé sur la taille du produit en cours d'adaptation et d'un modificateur qui calcule l'effort requis pour l'insertion dans le produit existant. Ce paramètre est appelé Modificateur de l'Ajustement de l'Adaptation (**AAM**).

AAM utilise, la compréhension du logiciel (SU), la Méconnaissance logicielle (UNFM), et de l'évaluation et l'assimilation (AA) avec un facteur appelé Facteur d'ajustement pour l'adaptation (**AAF**).

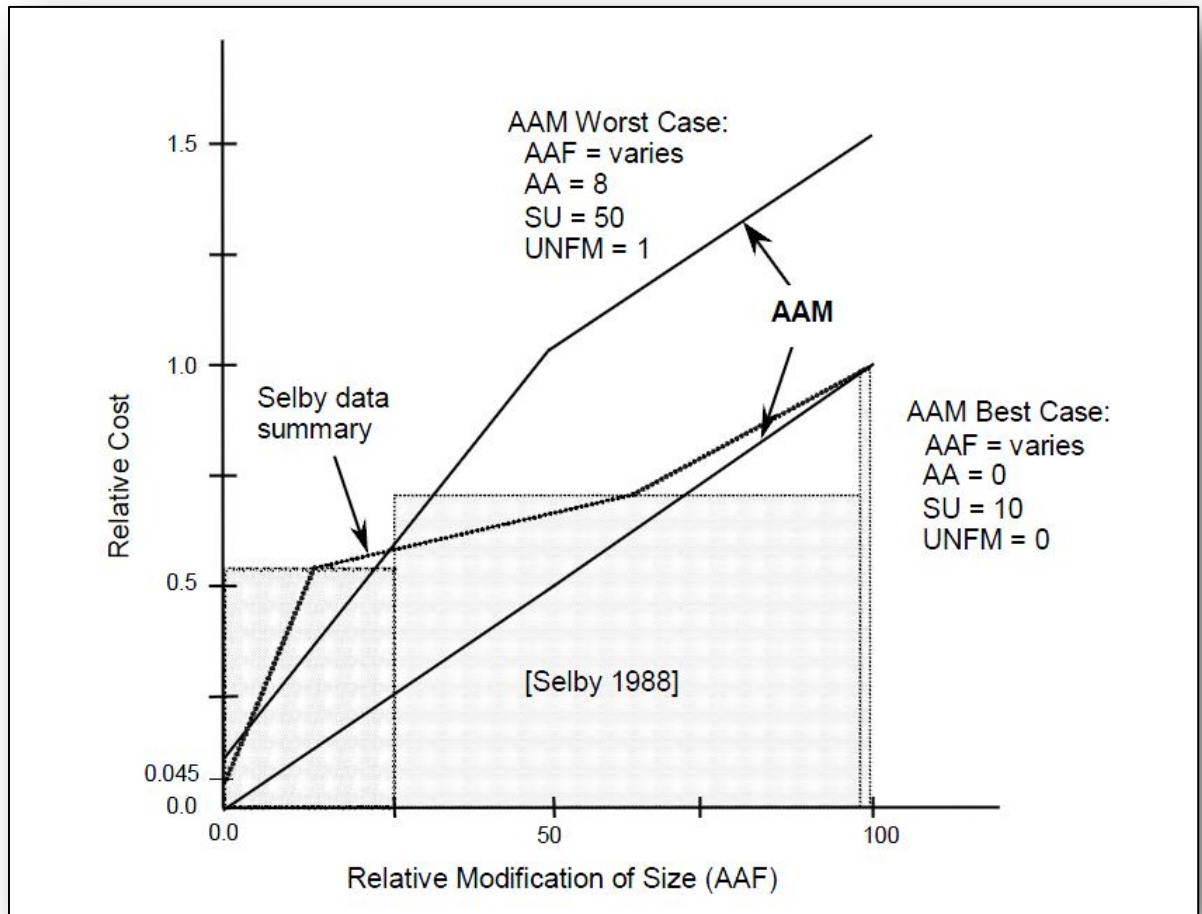
AAF contient des quantités de DM, CM, et IM où :

- **DM** (Pourcentage de modification du Design) est le pourcentage de la conception logicielle impactée par les modifications afin de l'accorder aux nouveaux objectifs et à l'environnement. (Ceci est nécessairement une quantification subjective.)
- **CM** (Code Pourcentage de modification) est le pourcentage de code du logiciel adapté, qui est modifié afin de l'adapter aux nouveaux objectifs et de l'environnement.
- **IM** (pourcentage d'intégration requis pour les logiciels adaptés) est le pourcentage de l'effort nécessaire pour intégrer le logiciel adapté dans un produit global et de tester le résultat. L'interprétation est effectuée par rapport à la quantité normale de l'intégration et des efforts de tests pour les logiciels de tailles comparables.

S'il n'y a pas de DM ou CM (le composant est utilisé non modifié), alors SU n'est pas nécessaire. Si le code est en cours de modification, alors SU s'applique.

La gamme de valeurs d'AAM est en partie donnée dans le graphe ci-dessous.

Dans le pire des cas, l'effort pour modifier un module réutilisable est deux fois plus grand que l'effort qu'il a fallu pour le développer (la valeur de l'AAM peut dépasser 100). Le cas le plus favorable se trouve lors d'une correspondance entre une adaptation du produit existant et le développement « from scratch ».



Relative Modification of Size (AAF)

4.2.2.5 Directives pour la quantification des logiciels adaptés

Cette section fournit des directives pour estimer les facteurs logiciels adaptés aux différentes catégories de code, en utilisant COCOMO II. Ce paragraphe se réfère à un logiciel développé « from scratch ». Le code adapté est un code préexistant sur lequel quelques modifications sont effectuées alors que le code réutilisé ne contient pas de changement par rapport aux sources préexistantes (utilisé tel quel). Les COTS, logiciels sur étagères, sont généralement traités de la même façon qu'un code réutilisé n'ayant à subir aucun changement. La différence entre les deux est que le COTS peut avoir besoin de « glue Code » pour l'intégrer au reste de l'application. Ce « glue code » doit être compté.

S'il n'y a pas de code source à modifié ou réutilisé, ni de COTS à intégrer, alors DM = 0, CM = 0, et SU et UNFM ne s'appliquent pas. AA et IM peuvent avoir des valeurs non nulles. Réutiliser ne veut pas dire sans effort d'intégration et tests. Cependant, dans l'approche de réutilisation, avec des produits bien architecturée, l'intégration et les tests sont minimales.

Pour les logiciels adaptés, CM > 0, DM est généralement > 0, et tous les autres facteurs de réutilisation ont normalement des valeurs non nulles. IM devrait être au moins modéré, mais peut être supérieur à 100% pour les applications plus complexes.

Le tableau montre les plages de validité des facteurs de réutilisation

Code des catégories	DM	CM	IM	AA	SU	UNFM
Nouveau Logiciel « From scratch »	NON	APPLICABLE	NON	APPLICABLE	NON	APPLICABLE
Adapté sur logiciel existant	0% - 100% Normalt. > 0%	0+% -100% généralt > DM et doit être > 0%	0% - 100+% IM généralt modérée et peut être > 100%	0% – 8%	0%-50%	0-1
Réutilisé Logiciel existant non changé	0%	0%	0% - 100% raremt 0%, mais doit être très petit	0% – 8%	NON	APPLICABLE
COTS Logiciel sur étagère nécessitant généralement une « glue code » pour l'intégrer	0%	0%	0% - 100%	0% – 8%	NON	APPLICABLE

4.3 EXIGENCES : EVOLUTION ET VOLATILITE

COCOMO II utilise un facteur appelé REVL, pour ajuster la taille effective du produit, en fonction de l'évolution des exigences et de leur volatilité, liées à des facteurs tels que l'évolution de la mission ou de l'interface utilisateur, la modernisation technologique, ou la volatilité COTS. Ce facteur représente le pourcentage de code éliminé du fait de l'évolution des exigences.

Par exemple : un projet qui comporte 100 000 instructions, mais dont les rejets sont équivalents à 20 000 instructions supplémentaires, a une valeur de 20 REVL. Ceci est utilisé pour ajuster la taille effective du projet à 120 000 instructions lors d'une estimation de COCOMO II.

L'utilisation de REVL dans le calcul de la taille logicielle est donnée par l'équation suivante :

$$\text{Taille} = \left(1 + \frac{\text{REVL}}{100}\right) \times \text{taille}_D$$

Taille_D est l'équivalence de la réutilisation (reuse-equivalent) du logiciel livré

4.4 REINGENIERIE ET CONVERSION DE CODE

COCOMO II utilise des ajouts au modèle pour intégrer les coûts liés à la réingénierie logicielle et la conversion. La différence majeure entre la réingénierie et la conversion réside en l'efficacité des outils automatisés pour restructurer le logiciel. Ceux-ci ont un taux de rentabilité très élevée : pour un code modifié (CM), très peu d'efforts sont nécessaires pour effectuer la réingénierie.

Par exemple : Dans l'étude de cas de réingénierie du NIST [Ruhl-Gunn 1991], 80% du code (13 131 déclarations en code source COBOL) a été repensé par « traduction » automatique. Il en a résulté un effort de réingénierie de 35 Homme/mois, soit un montant 4 fois inférieur à l'estimation de 152 Homme/Mois de COCOMO.

La méthode COCOMO II introduit pour les besoins d'estimation des charges de réingénierie et de conversion, un facteur supplémentaire, **AT**. AT est le pourcentage de code qui est repensé par le « traducteur » automatique. Basé sur une analyse des données du projet ci-dessus, la valeur par défaut de la productivité de la traduction automatique est de 2400 déclarations source par homme/mois. Cette valeur peut varier avec la technologie et est désigné dans le modèle COCOMO II par un autre facteur appelé **ATPROD**.

Dans l'exemple du NIST : ATPROD = 2400



L'équation montre comment la traduction automatique affecte l'effort estimé, PM_{Auto}

$$M_{Auto} = \frac{SLOC \text{ Adaptées} \times \left(\frac{AT}{100}\right)}{ATPROD}$$

L'étude de cas du NIST fournit aussi des indications utiles sur l'estimation du facteur AT. Ce facteur est intéressant puisqu'il permet de faire la différence de périmètre entre l'ancien code et le code remanié (par exemple utilisation de nouveaux COTS, changement d'un mode batch à un mode interactif,...).

Les données du NIST sur le pourcentage de traduction automatique (à l'origine: une application effectuant des traitements batchs sans intégration de COTS) sont donnés dans le tableau suivant [Ruhl-Gunn 1991].

Cible de réingénierie	AT (% automatique de traduction)
Batch processing	96%
Batch with SORT	90%
Batch with DBMS	88%
Batch, SORT, DBMS	82%
Interactive	50%

Variation du pourcentage de réingéniering automatique

La traduction automatique est considérée comme une activité distincte de développement. Ainsi, les SLOC Adaptés ne sont pas inclus dans la taille comptée en équivalent KSLOC, et de même son PM_{Auto} n'est pas inclus dans le PM_{NS} de l'estimation du projet. Si la traduction automatique des SLOC Adapté est incluse dans la taille en KSLOC équivalent, il doit être retiré pour éviter les doubles comptages. Cela se fait par l'ajout du terme $(1 - AT/100)$ à l'équation de KSLOC équivalent. ([voir équations de réutilisation](#))

4.5 CALCUL DE LA TAILLE DE LA MAINTENANCE LOGICIELLE

COCOMO II diffère de COCOMO 81. COCOMO II applique les facteurs d'échelle sur la taille du code modifié ; COCOMO81 les applique sur la taille du produit en cours de modification.

En appliquant les facteurs d'échelle à une application de 10 millions de SLOC, produit une estimation extrêmement haute, comme la plupart des estimations de produit n'était pas été affecté par des modifications. COCOMOII prend en compte les conséquences d'un produit en cours de modification, de part les facteurs de non compréhension logicielle et de non-familiarisation, discutés lors de la [réutilisation de composants](#)

La portée de la «maintenance du logiciel » suit les directives de COCOMO 81. Ceci comprend l'ajout, l'adaptation ou la fixation de fonctionnalités et exclut la reconstruction à plus de 50% des logiciels existants, et les développements importants (plus de 20% changé), les interfaces nécessitant peu de retouches sur le système existant.

La taille de maintenance est normalement obtenue par l'équation suivante, lorsque la taille du code de base est connue ainsi que le pourcentage de modification.

$$(\text{Size})_M = [(\text{Taille du code de base}) \times \text{MCF}] \times \text{MAF}$$

MAF : Facteur d'ajustement de la maintenance

MCF : Facteur de changement lors de la maintenance. MCF est un pourcentage

(Size)_M : taille du logiciel en maintenance. Cette taille est nécessaire lors de l'estimation des charges de maintenance.

MCF est similaire au « Annual Change traffic » de cocomo 81, excepté la période de maintenance qui peut être plus grande. Conceptuellement MCF représente le ratio suivant :

$$\text{MCF} = \frac{\text{Taille Ajoutée} + \text{Taille modifiée}}{\text{Taille de base du code}}$$

Une version plus simple peut être utilisée lorsque la fraction de code ajoutée ou modifiée au code de base, pendant la période de maintenance, est connue.



$$\text{(Size)}_M = (\text{taille ajoutée} + \text{taille modifiée}) \times \text{MAF}$$

Le facteur d'ajustement de maintenance (MAF), est utilisé pour ajuster la taille effective de maintenance pour prendre en compte la compréhension du logiciel (SU) et la méconnaissance du programmeur (UNFM), comme la réutilisation.

$$\text{MAF} = 1 + (\text{SU}/100 \times \text{UNFM})$$

ANNEXE A : PASSAGE DES POINTS DE FONCTION EN LIGNES DE CODE

Pour utiliser COCOMO, il est nécessaire de convertir les points de fonction non ajustés (UFP) en lignes de code source et ceci par rapport au langage de développement utilisé (Ada, C, C ++, Pascal, etc.)

COCOMO II propose une conversion à la fois pour les modèles Early Design and Post-Architecture. Le taux de conversion actuel indiqué dans le tableau est de [Jones 1996]. Les mises à jour de ces ratios de conversion ainsi que des ratios supplémentaires peuvent être trouvés sur

<http://www.scribd.com/doc/39703430/SPR-Programming-Language-Table>

Table 4. UFP to SLOC Conversion Ratios

Language	Default SLOC / UFP	Language	Default SLOC / UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
AI Shell	49	Modula 2	80
APL	32	Pascal	91
Assembly - Basic	320	PERL	27
Assembly - Macro	213	PowerBuilder	16
Basic - ANSI	64	Prolog	64
Basic - Compiled	91	Query – Default	13
Basic - Visual	32	Report Generator	80
C	128	Second Generation Language	107
C++	55	Simulation – Default	46
Cobol (ANSI 85)	91	Spreadsheet	6
Database – Default	40	Third Generation Language	80
Fifth Generation Language	4	Unix Shell Scripts	107
First Generation Language	320	USR_1	1
Forth	64	USR_2	1
Fortran 77	107	USR_3	1
Fortran 95	71	USR_4	1
Fourth Generation Language	20	USR_5	1
High Level Language	64	Visual Basic 5.0	29
HTML 3.0	15	Visual C++	34
Java	53		