3. GESTION DE PROJET

3. GESTION DE PROJET

Il n'est pas rare pour un projet logiciel de dépasser le délai estimé de 25 à 100%, c'est même à partir de ces constatations que la crise du logiciel est apparue… Aujourd'hui si quelques projets ont encore des dérapages parfois catastrophiques, certains ne dépassent les prévisions que de 5 à 10% seulement, le présent chapitre donne quelques indications pour parvenir à ce type de résultat..

Il est indispensable pour le client comme pour le fournisseur (chef de projet et management) d'estimer à l'avance la durée (calendrier) et le coût (effort) d'un projet logiciel. Il convient également d'en mesurer les risques en recensant les dépendances extérieures qui influeront sur les coûts et délais. Une remarque préalable s'impose quant au processus d'estimation : beaucoup trop souvent, le management et les clients ont de la peine à comprendre que l'estimation est une activité comportant des difficultés intrinsèques et la rendent encore plus difficile par ce manque de compréhension. Le processus requiert des raffinements successifs, il est impossible de fournir une estimation correcte jusqu'à ce que le logiciel soit appréhendé dans son intégralité.

L'estimation des coûts et durée se fait en trois étapes :

lors d'une réponse à un appel d'offres où il s'agit de fournir au plus vite une réponse adaptée au marché,

lors de la planification du projet où il s'agit d'établir le plan projet et le plan qualité qui serviront de cadre contractuel au projet,

lors du déroulement du projet afin d'affiner les prévisions et de les mettre à jour.

3.1. PROCESSUS D'ESTIMATION

La meilleure façon d'évaluer le montant à proposer dans un devis est sans doute de connaître le budget que le client est prêt à lui consacrer…Il est toutefois peu probable que le client ait la naïveté de faire part de ses prévisions de dépenses.

L'estimation du coût total d'un projet logiciel comprend le coût de développement du logiciel et du matériel, le coût de la formation, le coût des outils… Nous nous limitons ici au coût de développement du logiciel qui sera calculé en fonction du temps passé à développer celui-ci par les ingénieurs dont on connaît le tarif horaire et le taux d'implication dans le projet en terme de pourcentage de leur temps. Les coûts du développement logiciel sont donc étroitement liés à la notion de productivité des ingénieurs logiciels, c'est à dire le nombre de lignes de code (validées) produites par unité de temps. A noter qu'à ces coûts salariaux chargés, il ne faudra pas oublier de rajouter les coûts fixes selon un mode de calcul propre à chaque entreprise. Dans la suite lorsque nous parlerons de coûts, nous entendrons donc effort en terme d'hommes/mois.

Une méthode raisonnable pour établir une réponse à un appel d'offres, est de faire appel à des experts qui vont procéder par analogie avec des projets déjà achevés. On évalue chaque élément important du nouveau système en terme de pourcentage de coût d'un élément comparable dans le système achevé. L'estimation du coût total du nouveau projet est obtenue par sommation des estimations élémentaires. L'estimation est un exercice difficile, impossible en théorie et faisant donc appel à des heuristiques. Dans de trop nombreux cas, l'estimation préalable est faite à la hâte dans le but de remporter un marché, la planification qui s'ensuit est souvent pénalisée par une sous-estimation initiale des coûts et délais globaux.

Nous suggérons, en accord avec B.Boehm d'éviter de donner une estimation qui s'appuie sur un chiffre unique mais recommandons plutôt d'encadrer les prédictions par une fourchette optimiste/pessimiste. Le tableau ci dessous (Adapté *de "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0" , Bohem et al. 1995)* donne une base d'estimation pour passer d'un chiffre unique à une fourchette grâce à des coefficients multiplicateurs.

Phase	Taille et effort		durée	
	Optimiste	Pessimiste	Optimiste	Pessimiste
Analyse initiale des besoins	0.25	4.0	0.60	1.60
Définition approuvée des besoins	0.5	2.0	0.80	1.25
Spécification des besoins	0.67	1.5	0.85	1.15
Conception Globale	0.80	1.25	0.90	1.10
Conception détaillée	0.90	1.10	0.95	1.05

Tableau 3. 1 : Coefficients multiplicateurs pour chaque phase du projet

Pour utiliser les facteurs multiplicatifs du tableau, il suffit de multiplier le point d'estimation unique par les coefficients appropriés de manière à obtenir une fourchette d'estimation dont on peut remarquer qu'elle se

resserre au fur et à mesure que l'on avance dans les phases (0.90 à 1.10 en conception détaillée contre 0.25 à 4 en analyse initiale des besoins, donc au moment de la réponse à l'appel d'offres!)

Plutôt que d'estimer la durée du projet dans son ensemble on peut le décompose et estimer chacun de ses composants. Nous verrons que cette technique s'applique très bien à l'approche objets du fait de l'indépendance des différents objets entre eux.

Quelques règles d'or pour l'estimation

- **Eviter de deviner** : prendre le temps de réfléchir , ne jamais répondre sans avoir étudier la situation calmement
- **Prévoir du temps** pour l'estimation et la planifier
- Utiliser des données de **projets précédents**
- Prendre **l'avis des développeurs** qui vont effectivement effectuer le travail
- Estimer par **consensus** : consulter les différentes équipes , converger sur des dates au plus tôt et au plus tard
- Estimer par **difficultés** (facile, moyen, difficile)
- Ne pas oublier les **tâches récurrentes** : documentation, préparation des démonstrations utilisateurs, formation utilisateurs et développeurs, intégration à l'existant, récupération des données, revues, réunions, maintenance de l'existant pendant le développement du nouveau produit, gestion qualité, absentéisme (congés, maladie, RTT)
- Utiliser plusieurs techniques d'estimation
- Changer de méthodes d'estimation au fil de l'avancement du projet
- Prendre en compte les risques de gestion dans l'estimation

De manière générale, le processus d'estimation passe par 3 étapes

1. Estimer la taille du produit (nombre de lignes de code ou points de fonctions)
2. Estimer l'effort (en homme mois)
3. Estimer la durée (en mois ou semaines calendaires)

3.2. ESTIMATION DE LA TAILLE VIA LES POINTS DE FONCTION (ALBRECHT 79)

L'estimation des coûts passe comme nous l'avons vu en début de chapitre par l'estimation de la taille et de la productivité. Dans un premier temps nous avons défini la productivité en terme de nombre de lignes de code source produites par unité de temps, cette métrique simple présente cependant certains inconvénients:

- l'écriture de lignes de code représente une toute petite partie du développement
- pour des langages différents un même logiciel se code avec un nombre de lignes différent
- comment compter le nombre de lignes (commentaires)?
- on écrit souvent du code non livré (outils de développement)
- le nombre de lignes ne peut être compté qu'a posteriori

Un autre moyen de mesurer la productivité consiste à évaluer le nombre de fonctionnalités intéressantes produites par unités de temps. La méthode des points de fonction, définie chez IBM par Albrecht en 1979, puis rafinée en 83 propose ainsi d'estimer la taille d'un logiciel à partir de valeurs connues tôt dans le cycle de vie et qui sont indépendantes du langage choisi pour l'implémentation. La méthode est aujourd'hui défendue et mise à jour par l'IFPUG (the International Function Point User Group) qui publie the Function Point Counting Practices Manual dont la version courante est le IFPUG Manual 4.1.

Le nombre de points de fonctions d'un programme est basé sur le nombre et la complexité de cinq paramètres.:

Les données

Entrées externes (External Inputs EI) - Ces données peuvent venir d'un écran ou d'une autre application, elles peuvent être utilisées pour mettre à jour un ou plusieurs fichiers internes, il peut s'agir d'informations ou de données de contrôle. Les écrans, boites de dialogue, messages à travers lesquels un utilisateur entre une donnée font partie des entrées externes.

Sorties externes. (**External outputs EO) -** Ce sont les sorties d'un programme, elles peuvent prendre la forme de fichiers de sortie envoyés à d'autres application, d'écrans, de rapports de graphes destinés à l'utilisateur final ; ils sont créés à partir de fichiers logiques internes.

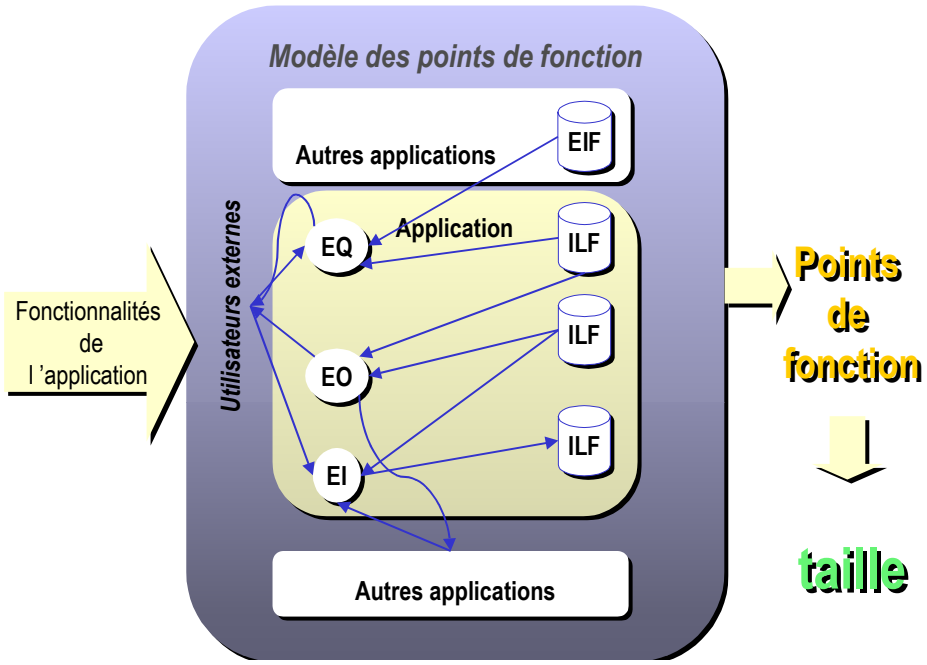

Figure 1 : Modèle des points de fonction

Les transactions

Requêtes externes (External Inquiry EQ) - Un procédé élémentaire mettant en jeu des entrées et des sorties qui résulte en la production de données provenant de plus d'un fichier logique interne. Le processus ne met pas à jour de fichier interne, combinaison d'entrées /sorties où le résultat apparaît sous forme simple et immédiate, généralement en utilisant une seule clé d'accès. La frontière entre requête externe et donnée externe est assez floue. Nous dirons que les requêtes concernent exclusivement les accès à une base de données alors que les données externes combinent ces accès avec des calculs et traitements plus ou moins complexes.

Fichiers logiques internes (Internal Logical Files ILF's) - Un groupe de données corrélées entièrement résidant dans les limites de l'application, mis à jour par les données externes, entièrement contrôlé par le programme. Par exemple un fichier, une table dans une base de données…

Fichier interfaces externes (External Interface Files EIF's) - Un groupe de données corrélées utilisées pour des références seulement. Les données résident entièrement hors de l'application et sont mises à jour par une autre application (c'est donc un ILF pour une autre application)

Après que les composants du projet aient été ainsi classifiés, chaque paramètre est affecté d'un score : faible, moyen, élevé. Pour les transactions (EI's, EO's, EQ's) le score est calculé sur le nombre de fichiers mis à jour ou référencés (FTR's) et le nombre de types de données impliquées (DET's).

FTR's	Données		
	1-4	5-15	>15
0-1	faible	faible	moyen
2	faible	moyen	haut
>2	moyen	haut	haut

Tableau 3.2: Table EI:

Une requête est notée (faible, moyenne ou haute) comme une sortie externe (EO) mais on lui attribue une valeur plutôt comme à une entrée externe. La notation est basée sur le nombre total des éléments de données et des types de fichiers référencés (combinaison "dédoublonnée" des entrées et sorties). Si un fichier ou un type est utilisé plusieurs fois il est répertorié une seule fois.

FTR's	Éléments de Données		
	1-5	6-19	>19
0-1	faible	faible	moyen
2-3	faible	moyen	haut
>3	moyen	haut	haut

Tableau 3.3 : Table EO et EQ:

Pour les fichiers externes et internes (ILF's and EIF's) le score (haut, moyen , élevé) est basé sur le type des éléments d' enregistrements (Record Element Type's) et le nombre de types des données (Data Element Type's). Un type d'élément d'enregistrement est un sous groupe reconnaissable de données dans un ILF ou EIF. Un type de donnée est un champ unique non récursif.

RET's	Data Elements		
	1-19	20-50	>50
1	faible	faible	moyen
2-5	faible	moyen	haut
>5	moyen	haut	haut

Tableau 3. 4Table ILF and EIF:

On récapitule ensuite dans un tableau les scores de chacun des paramètres et on obtient le nombre de points de fonction bruts (UFP)

Caractéristiques du programme	Points de fonction			
	Faible complexité	Complexité moyenne	Haute complexité	Total
Nombre d'entrées externes	__*3=	__*4=	__*6=	__
Nombre de sorties externes	__*4=	__*5=	__*7=	+__
Nombre de requêtes	__*3=	__*4=	__*6=	+__
Nombre de fichiers logiques internes	__*7=	__*10=	__*15=	+__
Nombre de fichiers interfaces externes	__*5=	__*7=	__*10=	+__
			Nombre total de points de fonctions bruts (UFP)	__
			Multiplié par le facteur d'ajustement* TCF	__
			Total de points de fonction ajustés AFP	__

Tableau 3.5 Multiplicateurs de points de fonction.
source: adapté de "Applied Software Measurement" (Jones 1991)

On calcule ensuite le facteur d'ajustement à partir de 14 caractéristiques générales du système. Pour chaque caractéristique on évalue le **degré d'influence** DI de 0 (aucune influence) à 5 (très grande influence). La table ci-dessous donne un aperçu des caractéristiques générales du système pour les auteurs du manuel utilisateurs de l'IFPUG.

Caractéristiques générales du système	**Description**
Communication de données	Combien de facilités de communication pour aider au transfert ou à l'échange d'information avec l'application ou le système?
Distribution du traitement et des données	Comment les données et les traitements distribués sont ils gérés
Critères de performance	L'utilisateur a t il des exigences en matière de temps de réponse?
Configuration matérielle très chargée	Quel est l'état de charge actuel de la plate-forme matérielle sur laquelle le système sera mis en exploitation?
Fréquence des transactions	Quelle est la fréquence d'exécution des transactions (quotidien, hebdomadaire, mensuel…)
Données saisies en temps réel	Quel est le pourcentage de données saisies en temps réel?
Efficacité des interfaces utilisateur	Les interfaces ont elles été dessinées pour atteindre à l'efficacité maximale de l'utilisateur
Mise à jour en temps réel des fichiers internes logiques	Combien de fichiers logiques internes sont ils mis à jour en temps réel?
Calculs complexes	L'application fait elle appel à des traitements logiques ou mathématiques complexes?
Réutilisabilité	L'application est elle développée pour satisfaire un ou plusieurs besoins clients?
Facilité d'installation	Quelle est la difficulté de conversion et d'installation?
Facilité opérationnelle	Quelle est l'efficacité et /ou l'automatisation des procédures de démarrage, backup, et récupération en cas de panne ?
Portabilité	L'application est elle spécifiquement conçue, développée maintenue pour être installée sur de multiples sites pour de multiples organisations?
Evolutivité	L'application est elle spécifiquement conçue, développée maintenue pour faciliter le changement?

Tableau 3.6 : Caractéristiques générales du système

Une fois que les caractéristiques générales du système ont été évaluées avec leur degré d'influence respectif, on peut calculer le **degré d'influence DI.** (0<DI<70) qui est la somme des degrés d'influence de chaque caractéristique.

Le facteur d'ajustement ou facteur technique de complexité se calcule par :

TCF = **0.65 + DI/100 , on a (0.65<TCF<1.35)**

Et enfin on calcule le nombre de points de fonctions ajustés:

FP = TCF * UFC

On peut maintenant calculer les coûts, l'effort et le calendrier à partir d'expériences précédentes, ou bien utiliser une méthode rapide de calcul du planning (voir ci dessous).

De nombreuses expériences ont montré la supériorité de l'évaluation par les points de fonction (Jones constate une marge d'erreur de 200% avec les points de fonction alors qu'elle est de 800% avec le nombre de lignes de code source!!.)

La méthode des points de fonction ne repose sur aucune technologie, il y a évidemment une relation entre le nombre de lignes de codes et le nombre de points de fonction que l'on peut trouver dans les archives de sa propre entreprise ou bien en utilisant les tables de conversion nombre de points de fonction / nombre de lignes de code que l'on peut trouver sur le site de l'IFPUG (table de correspondance répertoriant plus de 500 langages). Cette table permet de prédire le nombre de lignes assez tôt. Elle permet aussi d'étudier les productivités comparées lors de programmation dans différents langages et de convertir la taille d'une application dans n'importe quel langage.

Comme on peut le voir sur les caractéristique générales retenues pour un système, la méthode des points de fonctions a été définie pour des projets essentiellement orientés gestion. En 1986 la méthode dite des Features

points propose des extensions pour pouvoir être appliquée en contexte industriel (logiciels embarqués, télécoms, systèmes d'exploitation), elle prend en compte un nouveau paramètre, le nombre d'algorithmes. Sur les projets "systèmes d'information les deux méthodes convergent mais sur un petit système temps réel on a pu constater des divergences pouvant aller jusqu'à 45 points de fonction contre 70 *features points*.

3.3. ESTIMATIONS DE L'EFFORT: MODELE COCOMO (BOEHM, 1981, 1998)

L'estimation de l'effort (en homme mois) fait suite à l'estimation de la taille (en lignes de code source) et permettra de définir un calendrier pour le projet.

La méthode COCOMO fournit un algorithme permettant de dériver une évaluation, de l'effort et du planning à partir de l'estimation de la taille du logiciel. Nous donnons en référence la méthode de (Putman and Myers 1992) qui conduit également à une évaluation de l'effort

3.3.1 Description de la méthode

La méthode COCOMO, pour COnstructive COst Model a été développé par Dr. Barry Boehm pour estimer l'effort et le temps de développement d'un produit logiciel. A l'origine elle a été construite à partir d'une analyse des données par régression pratiquée sur 63 projets logiciels (gestion et informatique industrielle) comprenant de 2000 à 100.000 lignes de code dans l'entreprise TRW (USA). COCOMO à l'avantage d'être un modèle ouvert. Les données de calibrage, les formules et tous les détails des définitions sont disponibles. La participation à son développement est encouragée.

Aujourd'hui, COCOMO II est un nouveau produit beaucoup plus adapté à l'aspect réutilisation des composants (modules existants). La version 1998 a été calibrée sur 161 points de données, en utilisant l'approche statistique Bayesienne pour combiner les données empiriques avec les avis d'experts. De plus elle peut être re-calibrée sur les données de l'entreprise. Un nouveau modèle appelé COCOTS, est en cours de développement par l'équipe de COCOMO. Ce modèle permet d'estimer les coûts et planifier des projets logiciels utilisant des composants existants. C'est encore un projet de recherche.

Pour les projets basés sur une technologie traditionnelle, le modèle original de 1981 est encore valable, d'autant plus qu'il est maintenant rodé et bien documenté.

3.3.2 COCOMO 81

Le modèle COCOMO 81 s'appuie sur une estimation de l'effort en homme*mois (HM) calculée par la formule

Effort = C*M*tailles**

C facteur de complexité

M facteur multiplicateur

Taille en milliers de lignes de code source livrées (KDSI)

s proche de 1

Hypothèses :

KDSI *livré* :

- Exclut en général les environnements de tests, les supports de développement
- Exclut les commentaires
- Inclut le shell

HOMMES MOIS (HM)

- MM = 152 Heures (Normes américaines), tient compte des vacances, arrêts maladie...

Le modèle COCOMO 81 est en fait constitué de trois modèles :

- le modèle de base
- le modèle intermédiaire
- le modèle détaillé ou expert

Le modèle de base

Le modèle de base estime l'effort (le nombre de homme mois) en fonction du nombre de milliers d'instructions source livrées(*KDSI*), de la productivité (le nombre de lignes de code par personne par mois) et d'un facteur d'échelle qui dépend du type de projet.

Les 3 types de projet identifiés sont :

- organique :
  Il est défini par une innovation minimale, une organisation simple en petites équipes expérimentées. (ex: petite gestion, système de notes dans une école , petits systèmes d'exploitation, traducteurs)

- médian (*semi-detached*) :
  Il est défini par un degré d'innovation raisonnable, (exemples: Compilateurs, Contrôle de processus simple, système bancaire interactif)
- imbriqué
  Dans ces projets le niveau d'innovation est important , l'organisation complexe, couplage fort avec beaucoup d'interactions, (exemples: Gros système d'exploitation, Systèmes transactionnels complexes, système de contrôle aérospatial..)

TYPE DE PROJET	effort en homme mois (HM)	Temps de développement
ORGANIQUE	$2.4(KDSI)^{1.05}$	$2.5(HM)^{0.38}$
MEDIAN	$3.0(KDSI)^{1.12}$	$2.5(HM)^{0.35}$
IMBRIQUE	$3.6(KDSI)^{1.20}$	$2.5(HM)^{0.32}$

Tableau 3. 7 formules d'estimation COCOMO pour le modèle de base

EXEMPLES

Effort (HM)	2KDSI	8KDSI	32KDSI	128 KDSI	512 KDSI
Organique	5	21,3	91	392	
Médian	6,5	31	146	687	3250
Imbriqué	8,3	44	230	1216	6420

Tableau 3. 8 : Effort en HOMMES MOIS (HM) en fonction de la taille et du type de projet

TDEV(mois)	2KDSI	8KDSI	32KDSI	128 KDSI	512 KDSI
Organique	4.6	8	14	24	
Médian	4.8	8,3	14	24	42
Imbriqué	4.9	8,4	14	24	41

Tableau 3.9 : Temps de développement (en mois) en fonction de la taille et du type de projet

Le temps de développement commence après les spécifications fonctionnelles et s'arrête après l'intégration
De ces chiffres on peut déduire la productivité (KDSI/HM) et le nombre moyen de personnes sur le projet (FSP=HM/TDEV).
On peut ensuite calculer la distribution de l'effort par phases (en %)

- RPD (Requirements and Preliminary Design): Conception globale et Plan d'intégration
- DD (Detail Design) : Conception détaillée
- CUT (Code and Unit Test) : Programmation et Tests unitaires
- IT (Integration and Test) : Intégration

PROJET ORGANIQUE	**2 KDSI**	**8 KDSI**	**32 KDSI**	**128 KDI**	**512 KDSI**
RPD	**16**	**16**	**16**	**16**	
DD	**26**	**25**	**24**	**23**	
CUT	**42**	**40**	**38**	**36**	
IT	**16**	**19**	**22**	**25**	
PROJET MEDIAN					
RPD	**17**	**17**	**17**	**17**	**17**
DD	**27**	**26**	**25**	**24**	**23**
CUT	**37**	**35**	**33**	**31**	**29**
IT	**19**	**22**	**25**	**28**	**31**
PROJET IMBRIQUE					
RPD	**18**	**18**	**18**	**18**	**18**
DD	**28**	**27**	**26**	**25**	**24**
CUT	**32**	**30**	**28**	**26**	**24**
IT	**22**	**25**	**28**	**31**	**34**

Tableau 3. 10 : distribution de l'effort par phases en pourcentatge

ORGANIQUE	2 KDSI	8 KDSI	32 KDSI	128 KDI	512 KDSI
RPD	19	19	19	19	
DD et CUT	63	59	55	51	
IT	18	22	26	30	
MEDIAN					
RPD	24	25	26	27	28
DD et CUT	56	52	48	44	40
IT	20	23	26	29	32
IMBRIQUE					
RPD	30	32	34	36	38
DD et CUT	48	44	40	36	32
IT	22	24	26	28	30

Tableau 3.11 distribution du temps de développement par phases en pourcentage

EXEMPLE : Soit un projet estimé à 32 *KDSI* en mode organique,

HM $= 2.4 * (32)^{1.05} = 91$ HM

TDEV $= 2.5 * (91)^{0.38} = 14$ Mois

PRODUCTIVITE = 32000 DSI/91 HM = 352 DSI/HM

FSP = 91 HM / 14 MOIS = 6.5 FSP

Conformément au tableau 3.10, on a

PHASE DE CONCEPTION : 0.16 * 91 = 14.5 HM

PHASE DE CODAGE : 0.62 * 91 = 56.5 HM

PHASE D'INTÉGRATION : 0.22 * 91 = 20 HM

Conformément au tableau 3.11, on a :

PHASE DE CONCEPTION : 0.19 * 14 = 2.6 Mois

PHASE DE CODAGE : 0.55 * 14 = 7.7 Mois

PHASE D'INTÉGRATION : 0.26 * 14 = 3.7 Mois

En divisant on obtient le nombre de personnes nécessaires pour chaque phase.

Le modèle intermédiaire

Le modèle de base ne prend en compte que le nombre de lignes source et induit des discontinuités un peu brutales au niveau du personnel entre chaque phase du cycle de vie ; ce qui peut perturber l'organisation du projet

Le modèle intermédiaire introduit 15 facteurs de productivité (appelés *'cost drivers'*), représentants un avis subjectif du produit, du matériel, du personnel, et des attributs du projet. Chaque facteur prend une valeur nominative de 1, et peut varier selon son importance dans le projet. Ces facteurs sont à rapprocher des caractéristiques générales du produit vus plus haut dans le paragraphe sur les points de fonction. Les 15 facteurs sont multipliés entre eux pour donner un facteur d'ajustement qui vient modifier l'estimation donnée par la formule de base.

Facteurs de productivité

- Logiciel
  - *RELY*: Fiabilité requise
  - *DATA*: Volume des données manipulées
  - *CPLX*: Complexité du produit
- Matériel
  - *TIME*: Contraintes de temps d'exécution
  - *STOR*: Contraintes de taille mémoire
  - *VIRT*: Instabilité de la mémoire
- Personnel
  - *ECAP*: Aptitude de l'équipe
  - *AEXP*: Expérience du domaine
  - *VEXP*: Expérience de la machine virtuelle
  - *LEXP*: Maîtrise du langage
- Projet

- *MODP*: Pratique de développement évoluées
- *TOOL*: Utilisation d'outils logiciels
- *SCED*: Contraintes de délais

Multiplicateurs	Très bas	Bas	Nominal	Elevé	Très élevé	Extrêmement élev
RELY	0,75	0,88	1	1,15	1,4	
DATA		0,94	1	1,08	1,16	
CPLX	0,7	0,85	1	1,15	1,3	1,65
TIME			1	1,11	1,3	1,66
STOR			1	1,06	1,21	1,56
VIRT		0,87	1	1,15	1,3	
ECAP	1,44	1,18	1	0,86	0,7	
AEXP	1,29	1,13	1	0,91	0,82	
VEXP	1,21	1,1	1	0,9		
LEXP	1,14	1,07	1	0,95		
MODP	1,24	1,1	1	0,91	0,82	
TOOL	1,24	1,1	1	0,91	0,83	
SCED	1,23	1,08	1	1,04	1,1	

Tableau 3.12 Coefficients multiplicateurs pour chacun des facteurs de productivité

TYPE DE PROJET	effort en homme mois (HM)	Temps de développement
ORGANIQUE	$3.2(KDSI)^{1.05}$	$2.5(HM)^{0.38}$
MEDIAN	$3.0(KDSI)^{1.12}$	$2.5(HM)^{0.35}$
IMBRIQUE	$2.8(KDSI)^{1.20}$	$2.5(HM)^{0.32}$

Tableau 3. 13 : coefficients pour le calcul d'effort et de temps dans le modèle intermédiaire

Calcul de l'effort (HM) dans le modèle intermédiaire

Identifier le mode de développement - organique, médian, imbriqué) . Ceci donne 4 coefficients au modèle : p (productivité nominative), e (échelle appliquée à la taille du logiciel), c (constante du modèle) et d (échelle appliquée au temps de développement) conformément au tableau 3.13.

Estimer le nombre de lignes du code source (en KDSI), puis calculer le nombre d'homme*mois par la formule appropriée du tableau 3.13. On obtient HM base

Estimer les 15 facteurs de productivité et calculer le facteur d'ajustement (a) en multipliant les facteurs ensemble conformément au tableau 3.12

Multiplier l'effort 'nominal' par le facteur d'ajustement : HM = HM base * a

Calculer le temps de développement en utilisant le tableau 3.13: TDEV = c(HH)^d (on notera que les coefficients restent inchangés par rapport au modèle de base)

Dans le modèle intermédiaire, les coefficients multiplicateurs s'appliquent uniformément sur l'ensemble des phases.

Modèle expert

Le modèle expert inclut toutes les caractéristiques du modèle intermédiaire avec une estimation de l'impact de la conduite des coûts sur chaque étape du cycle de développement : définition initiale du produit, définition détaillée, codage, intégration . De plus, le projet est analysé en terme d'une hiérarchie : module, sous système et système. Il permet une véritable gestion de projet, utile pour de grands projets. Le modèle expert n'est pas décrit ici.

3.3.2 COCOMO II

A la lecture du paragraphe précédent sur COCOMO 81, plusieurs questions viennent à l'esprit:

- une seule entreprise est-elle représentative comme base de développement de COCOMO?
- COCOMO reste très lié au nombre de lignes de code, surtout le modèle de base, mais plus les programmeurs sont experts (et leur salaire élevé), moins ils écrivent de lignes de code pour un même projet!
- Comment prendre en compte le réutilisation?

Afin de prendre en compte ces critiques, COCOMO II (Boehm 98)est apparue.

COCOMO II peut être calibré pour mieux correspondre aux projets de l'entreprise. COCOMO II tient compte de la réutilisation

COCOMO II est constitué de trois modèles :

Modèle de composition d'application :

Ce modèle est utilisé pour les projets fabriqués à l'aide des toolkits d'outils graphiques. Il est basé sur les nouveaux 'Object Points'.

Modèle avant projet :

Modèle utilisé pour obtenir une estimation approximative avant de connaître l'architecture définitive. Il utilise un sous ensemble de facteurs de productivité (cost drivers). Il est basé sur le nombre de lignes de code ou les points de fonction non ajustés.

Modèle post-architecture :

Il s'agit du modèle le plus détaillé de COCOMO II. A utiliser après le développement de l'architecture générale du projet. Il utilise des facteurs de productivité (cost drivers) et des formules.

Les facteurs utilisés sont classés en : Facteurs d'échelle, Urgence, Flexibilité de développement, résolution d'architecture/Risque, Cohésion d'équipe et Maturité de Processus.

3.3.3 Conclusion

COCOMO reste le plus connu des modèles d'estimation de coût de projet logiciel. Mais le modèle d' origine n'adresse pas les projets utilisant les composants logiciel existants. COCOMO II répond à ce problème, mais est encore jeune, Le projet de recherche appelé COCOTS est à suivre pour les personnes désirant estimer un projet logiciel moderne.

COCOMO reste la référence en matière d'estimation détaillée des coûts et surtout de la ventilation de ces coûts suivant les phases des projets. Les estimations de COCOMO sont d'autant plus fiables, que les paramètres du projet sont bien connus, c'est-à-dire qu'on a profité des projets précédents pour étalonner ces paramètres.

La principale faiblesse réside dans la nécessité d'avoir une estimation du nombre de lignes du logiciel. Cette taille du logiciel n'est connue qu'à la fin de la réalisation et le problème de son estimation reste entier même si l'approche par les points de fonction peut fournir une aide. Ainsi, le management devra évaluer la justesse des prévisions tout au long de la durée de vie du projet.

Les résultats montrent que les valeurs réelles sont égales à 20% près aux valeurs estimées par COCOMO dans 68 % des cas .