

CHAPITRE 5 :

Estimation des charges, délais et coûts

1. Introduction

Estimation

Estimer le coût d'un projet consiste à établir, au plus tard avant le début de sa mise en œuvre, un pronostic sur ce que devrait être son coût final. Plus d'une dizaine de méthodes d'estimation existent, les plus connues figurent dans ce lexique aux entrées suivantes :

- Estimation analytique,
- Méthode du prix gagnant,
- Estimation "à dire d'expert",
- Méthode Delphes,
- Méthode des barèmes,
- Planning
- poker,
- Diebold,
- CoCoMo,
- Méthode des points de fonction,
- Modèle de Freiman,
- Méthode Paramétrique, Méthode Analogique.

1. Introduction

Estimation

Hypothèse faite sur un résultat quantitatif et comportant une indication sur son exactitude (*FD X50-115*).

Note 1 : Dans le domaine du management de projet, l'hypothèse porte généralement sur les **coûts** ou les **délais**.

Note 2 : La précision des estimations varie selon les **phases** du projet.

Note 3 : Le coût de l'estimation varie ainsi que les **méthodes d'estimation**.

1. Introduction

Pourquoi estimer ?

- Montrer notre capacité à livrer une solution
- Evaluer le coût de la solution
- Evaluer les ressources nécessaires
- Se positionner face à la concurrence
- Répondre aux attentes du client

Quand estimer ?

- En avant vente pour qualifier une opportunité
- En production (en début de projet pour la mise en place d'une solution, à chaque changement de phase, lors d'un changement de périmètre du projet)
- En fin de projet (bilan).

1. Introduction

BESOIN D'ESTIMATION

- 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%.
- Il est indispensable pour le client comme pour le fournisseur 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 influenceront sur les coûts et délais.

1. Introduction

ESTIMATEUR

Estimateur (Estimator) : Personne chargée d'effectuer l'estimation des coûts d'un projet.

- Note 1 : Chez un maître d'ouvrage, l'estimateur devra évaluer des coûts d'exploitation et de fonctionnement, tout autant que des coûts d'investissement, mais avec des approximations assez larges.
- Note 2 : Chez un maître d'oeuvre ou un entrepreneur, l'estimateur devra surtout donner une évaluation raisonnable des risques financiers que l'entreprise va prendre ou non.

1. Introduction

DIVERSES ESTIMATIONS

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

- lors d'une réponse à un appel d'ordres 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.

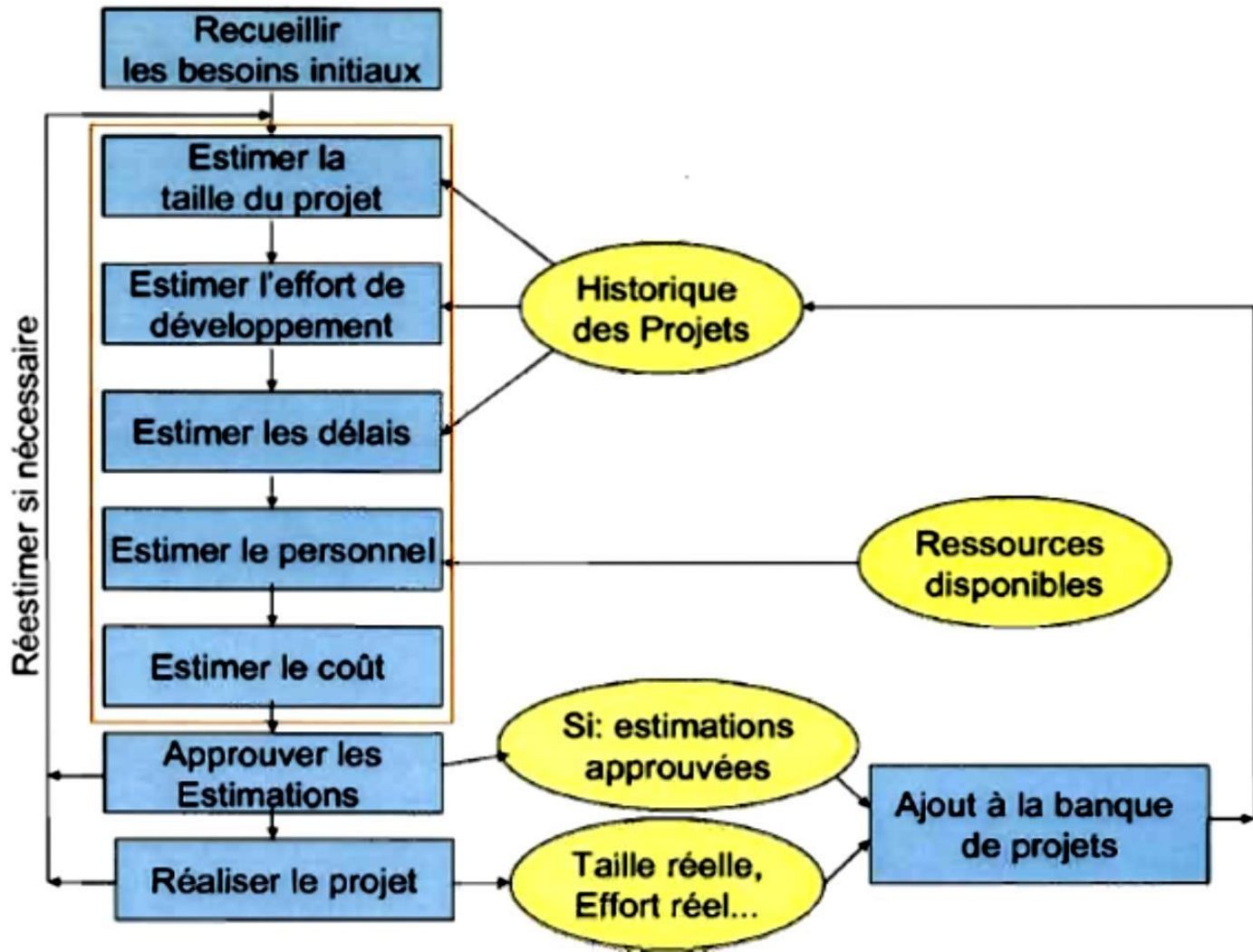
1. Introduction

LE PROCESSUS D'ESTIMATION

le processus d'estimation des projets informatiques comprend cinq étapes fondamentales:

- L'estimation de la taille.
- L'estimation de l'effort.
- L'estimation de la durée.
- L'estimation du personnel.
- L'estimation du coût.

1. Introduction



LES NON-MÉTHODES

- La méthode dite de Parkinson n'a rien à voir avec la maladie du même nom .
 - Elle fait référence à un auteur, historien et militaire, qui a traduit des observations récurrentes d'un dysfonctionnement administratif sous forme d'une « loi » .
- « Le travail se dilate jusqu'à remplir le temps disponible ».

1. Introduction

Loi de Parkinson

On peut résumer la loi de Parkinson comme suit : "le temps passé à accomplir une tâche intellectuelle s'adapte automatiquement au temps disponible". Il ne s'agit pas là d'un comportement malhonnête des ressources en charge de la tâche considérée, mais au contraire le plus souvent de perfectionnisme.

Conséquences

de ce phénomène :

- 1- Les marges prévues sont consommées et ne sont donc plus disponibles pour prévenir les aléas.
- 2- On constate un surcoût, en général sans amélioration notable de la qualité de livrable.

LES NON-MÉTHODES

Il existe cinq familles de méthodes d'estimation, qui ne sont pas forcément concurrentes, mais peuvent être utilisées simultanément, ou à des moments différents du cycle de vie du projet :

- Le jugement d'expert.
- L'estimation par analogie.
- L'estimation ascendante.
- L'estimation paramétrique.
- L'estimation probabiliste.

LE JUGEMENT D'EXPERT

- Certaines estimations sont basées sur Les experts sont ,dans le cas des projets système d'information, des consultants ou d'autres chefs de projet expérimentés, qui font appel à leurs connaissances explicites ou implicites ,acquises au cours de leur vécu et leurs observations.
- Estimation "à dire d'expert" :Les méthodes d'estimation des coûts "à dire d'expert" consistent à interroger un expert au moins du domaine. La fiabilité de l'estimation dépend de l'expérience de l'expert et de son impartialité.
- La méthode Delphi a formalisé un processus de mise en commun d'expertise.
- Méthode Delphes (Delphi) : La méthode Delphes est la plus connue des méthodes à dire d'expert. La méthode Delphes consiste à interroger plusieurs experts, en recherchant la convergence de leurs estimations individuelles.

1. Introduction

L'ESTIMATION PAR ANALOGIE

- Les estimations s'appuient sur les consommations de projets similaires pour lesquels on a conservé une mémoire. Ces projets peuvent être internes ou externes à l'entreprise.
- Pour estimer par analogie, il faut conserver les données de tous les projets antérieurs qui sont jugées pertinentes pour le dimensionnement d'un projet afin que, par comparaison, on évalue les caractéristiques du prochain projet. C'est la méthode « intuitive ».

1. Introduction

L'ESTIMATION PAR ANALOGIE

Méthode Analogique d'estimation des coûts

La Méthode Analogique est une méthode d'estimation des coûts basée sur le retour d'expérience des projets passés. Son principe consiste à effectuer une extrapolation à partir d'un nombre aussi important que possible de cas passés (cas source) similaires au cas présent (cas cible). Les cas source sont extraits de

la base de données historique des projets.

Deux méthodes pouvant se rattacher à cette famille :

- la méthode de répartition proportionnelle: elle est basée sur l'hypothèse que les différentes phases d'un cycle sont liés par une relation de proportionnalité.
- la méthode des ratios : elle s'appuie sur l'observation de pourcentages stables entre différentes activités.

1. Introduction

L'ESTIMATION ASCENDANTE

- Le principe de cette famille de méthodes est d'estimer la charge d'éléments de travail (lot de travail, activité, tâche...) et de totaliser ces estimations élémentaires.
- Elle requiert donc d'avoir réalisé une structure de découpage du projet suffisamment fine. La méthode d'évaluation analytique est fréquemment utilisée.

L'ESTIMATION PARAMÉTRIQUE

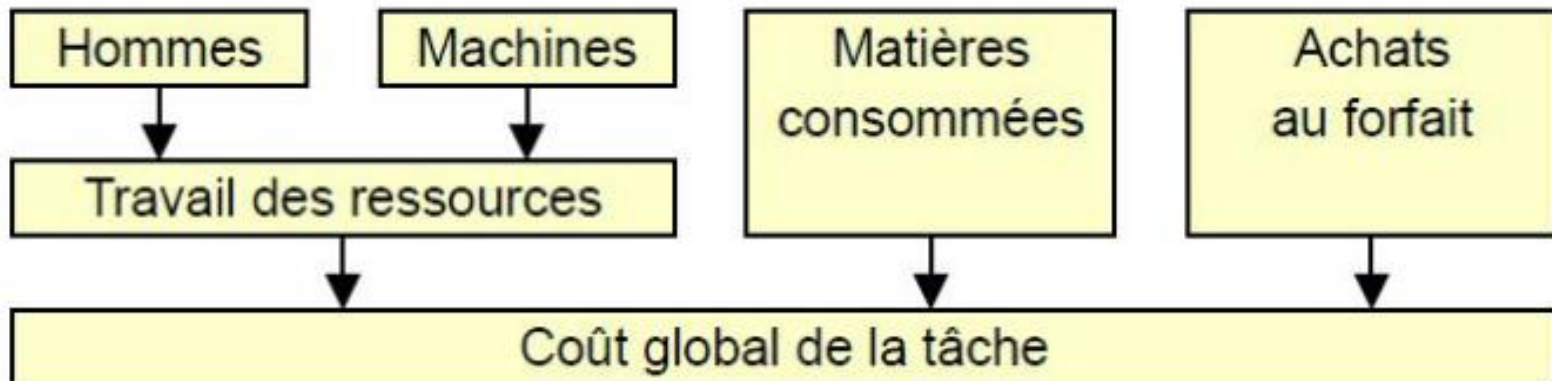
- Certaines méthodes utilisent un modèle mathématique, issu d'analyses statistiques sur la relation entre un ou plusieurs paramètres, qui sont des variables caractéristiques du projet, et la charge prévisible.
- L'estimation paramétrique est analogue à la technique des unités d'oeuvre utilisée en comptabilité analytique pour répartir des coûts généraux de façon simple, facile à calculer, uniforme (la même règle s'applique à tous), et pas trop fautive dans le sens où le résultat n'est pas trop éloigné de celui que l'on obtiendrait si l'on se livrait à un calcul direct et détaillé.

Pour utiliser cette technique, il faut déterminer une ou plusieurs unités d'oeuvre pertinentes, affecter un coût unitaire standard à chaque unité d'oeuvre (par exemple, en divisant un coût global précédemment mesuré par le nombre total d'unités d'oeuvre) et pouvoir dénombrer les unités d'oeuvre. Les méthodes basées sur un modèle proposent donc de repérer certains éléments simples, à partir desquels on calculera l'effort nécessaire. L'estimation paramétrique englobe deux méthodes, diversement utilisées : Le modèle COCOMO et La méthode des points de fonction.

1. Introduction

Estimation analytique des coûts

La méthode analytique d'estimation des coûts est la méthode la plus répandue et la plus précise. Elle est systématiquement utilisée lors de la phase de planification du projet. Le chiffrage ainsi obtenu sert à l'élaboration du référentiel des coûts (budget de référence). Son principal défaut est de nécessiter une connaissance fine du contenu du projet, ce qui exclut de pouvoir l'utiliser dans les phases très amont du projet (phase exploratoire).



1. Introduction

L'ESTIMATION PROBABILISTE

cette famille regroupe des approches utilisées conjointement avec d'autres méthodes lorsque l'incertitude est élevée et que l'on cherche à réduire le risque d'une évaluation erronée. L'estimation probabiliste englobe deux méthodes :

- **L'ESTIMATION À TROIS POINTS .**
- **LA MÉTHODE DE MONTE-CARLO.**
- **Méthode des points de fonction :** La méthode des points de fonction est une méthode d'estimation des coûts dédiée aux projets informatiques, la méthode des points de fonction est plus récente que Cocomo et Diebold. Elle présente l'avantage de se satisfaire comme donnée d'entrée de la connaissance des fonctions attendues de l'application.
- **La méthode de Monte-Carlo :** est une approche probabiliste de la durée ou du coût d'un projet. La méthode de Monte-Carlo est basée sur un calcul itératif prenant comme donnée d'entrée la valeur de durée (ou de coût) de chaque tâche tirée au hasard sur la base de la loi de probabilité propre à cette tâche.

2. La méthode COCOMO

- La méthode COCOMO, pour COConstructive COst MOdel a été développée 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).
- Aujourd'hui, COCOMO II est un nouveau produit beaucoup plus adapté à l'aspect réutilisation des composants (modules existants).

2. La méthode COCOMO - suite

Les modèles de COCOMO

1. 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(KISL), 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.

2. Le modèle intermédiaire

Le modèle intermédiaire reprend l'effort et la durée du modèle de base en appliquant cette fois-ci 15 facteurs de productivité (appelés '*cost drivers*'), représentant 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.

3. Modèle expert « détaillé »

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.

2. La méthode COCOMO - suite

Types de projets

1. 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*). Cette première catégorie regroupe les projets *allant jusqu'à 50 KDIS* (Delivered Source Instructions).

2. médian (semi-detached): Il est défini par un degré d'innovation raisonnable, (*exemples: Compilateurs, Contrôle de processus simple, système bancaire interactif*).

Cette catégorie regroupe les projets comporte *entre 50 000 et 300 000 instructions*.

3. 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*). Cette catégorie regroupe les projets comporte *plus de 300 000 instructions* et si l'on prévoit une équipe nombreuse ; il s'applique souvent à un domaine nouveau.

2. La méthode COCOMO - suite

Les modèles de COCOMO

1. Le modèle de base :

a. identifier le mode de développement : **organique, médian ou imbriqué**

Les formules :

$Effort = a * (KISL)^b$ exprimé en homme mois

$TDEV = 2.5 * (effort)^c$ exprimé en mois

De ces chiffres on peut déduire la **productivité (KDSI/HM)** et **le nombre moyen de personnes** sur le projet (**$FSP=HM/TDEV$**).

Boehm a déterminé les valeurs a, b et c à partir des résultats d'analyse statistique sur un échantillon significatif de projets

<u>TYPE DE PROJET</u>	<u>effort en homme mois (HM)</u>	<u>Temps de développement</u>
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}

2. La méthode COCOMO - suite

Exemple: Temps de développement (en mois) 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

Le temps de développement commence après les spécifications fonctionnelles et s'arrête après l'intégration.

b. distribution par phase

Distribution par phase, aussi appelé Ventilation, permet de déterminer le temps de développement et l'effort nécessaires pour chaque phase de cycle de développement. COCOMO divise le cycle de développement en 4 grandes phases :

- **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

Selon la complexité et la taille de l'application, l'effort et le temps varient sur chacune des phases ; COCOMO exprime cela sous la forme d'un coefficient représentant le pourcentage d'effort à réaliser et le temps passé.

2. La méthode COCOMO - suite

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 1: *Distribution de l'effort par phases en pourcentage*

2. La méthode COCOMO - suite

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 2: *distribution du temps de développement par phases en pourcentage*

2. La méthode COCOMO - suite

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

HM =	$2.4 * (32)^{1.05} = 91 \text{ HM}$
TDEV =	$2.5 * (91)^{0.38} = 14 \text{ Mois}$
PRODUCTIVITE =	$32000 \text{ DSI}/91 \text{ HM} = 352 \text{ DSI}/\text{HM}$
FSP =	$91 \text{ HM} / 14 \text{ MOIS} = 6.5 \text{ FSP}$

Conformément au tableau 1, on a

PHASE DE CONCEPTION : $0.16 * 91 = 14.5 \text{ HM}$

PHASE DE CODAGE : $0.62 * 91 = 56.5 \text{ HM}$

PHASE D'INTÉGRATION : $0.22 * 91 = 20 \text{ HM}$

Conformément au tableau 2, on a :

PHASE DE CONCEPTION : $0.19 * 14 = 2.6 \text{ Mois}$

PHASE DE CODAGE : $0.55 * 14 = 7.7 \text{ Mois}$

PHASE D'INTÉGRATION : $0.26 * 14 = 3.7 \text{ Mois}$

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

2. La méthode COCOMO - suite

2. Le modèle intermédiaire

- Le modèle intermédiaire introduit 15 facteurs de productivité (appelés '*cost drivers*'), représentant 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.
- Les 15 facteurs sont multipliés entre eux pour donner un facteur d'ajustement qui vient modifier l'estimation donnée par la formule de base.

<i>Logiciel</i>	- RELY : Fiabilité requise – DATA : Taille de la base de données - CPLX : Complexité du produit
<i>Matériel</i>	- TIME : Contraintes de temps d'exécution - STOR : Contraintes de taille mémoire principale - VIRT : instabilité de la machine virtuelle – TURN : Temps de Retournement
<i>Personnel</i>	- ACAP : compétence des analystes - AEXP : Expérience du domaine d'application - PCAP : Compétence des programmeurs - VEXP : Expérience de la machine virtuelle - LEXP : Expérience du langage
<i>Projet</i>	- MODP : Pratique des méthodes modernes - TOOL : Utilisation d'outils logiciels - SCED : Contraintes de planning

2. La méthode COCOMO - suite

COCOMO Intermédiaire Facteurs d'Ajustement		Très Bas	Bas	Normal	Haut	Très Haut	Extr. Haut
Attributs du produit							
Fiabilité Requise	RELY	0,75	0,88	1,00	1,15	1,40	
Taille de la Base de Données	DATA		0,94	1,00	1,08	1,16	
Complexité du produit	CPLX	0,70	0,85	1,00	1,15	1,30	1,65
Attributs du Matériel							
Contraintes de temps d'exécution	TIME			1,00	1,11	1,30	1,66
Contraintes de taille mémoire principale	STOR			1,00	1,06	1,21	1,56
Instabilité de la Machine Virtuelle	VIRT		0,87	1,00	1,15	1,30	
Temps de Retournement	TURN		0,87	1,00	1,07	1,15	
Attributs de l'équipe							
Compétence des Analystes	ACAP	1,46	1,19	1,00	0,86	0,71	
Expérience du domaine d'application	AEXP	1,29	1,13	1,00	0,91	0,82	
Compétence des Programmeurs	PCAP	1,42	1,17	1,00	0,86	0,70	
Expérience de la Machine Virtuelle	VEXP	1,21	1,10	1,00	0,90		
Expérience du langage	LEXP	1,14	1,07	1,00	0,95		
Méthodes et Outils							
Pratique des Méthodes Modernes	MODP	1,24	1,10	1,00	0,91	0,82	
Utilisation d'outils logiciels	TOOL	1,24	1,10	1,00	0,91	0,83	
Contraintes de planning	SCED	1,23	1,08	1,00	1,04	1,10	

2. La méthode COCOMO - suite

Ensuite on calcule le facteur d'ajustement de la charge par la formule suivante :

$$F = F_1 \times F_2 \times \dots \times F_{15}$$

- Si $F > 1$ la charge augmente
- Si $F = 1$ la charge n'est pas modifiée
- Si $F < 1$ la charge diminue

$$\text{Effort ajusté} = \text{effort} * F$$

$$T_{dev} = c * (\text{effort ajusté})^d$$

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

2. La méthode COCOMO - suite

Exemples : Considérons une situation où le COCOMO de base prévoit un effort de 45HM pour développer un système embarqué, le matériel est constitué d'un microprocesseur de 1,6 GH et d'une mémoire centrale de 512 MO, les multiplicateurs du modèle COCOMO intermédiaire ont tous une valeur moyenne à l'exception des suivants : **RELY= 1.15 TIME=1.21 STOR=1.10 TOOL=1.10**

Nous obtenons effort = **$45 \times 1.15 \times 1.21 \times 1.10 \times 1.10 = 76 \text{ HM}$**

Etant donné que le cout mensuel d'un ingénieur est de 20000 DA le cout total pour le développement pour ce projet est de : **cout = 76 x 20000=1520000 DA**

A l'examen de ces résultats, il est clair que les contraintes matérielles affectent de façon significative les couts de développement.

Supposons maintenant que nous utilisons un processeur **1,6 dual core** avec une RAM de **1 GO** ; les attributs **TIME** et **STORE** prennent alors une valeur moyenne le cout du logiciel devient :

COUT= 45 x 1.10x 1.15 x 20000=1138500 DA d'où une économie de **381500 DA**
Obtenue en achetant du matériel plus performant.

2. La méthode COCOMO - suite

Avantages & Inconvénients

Les avantages :

- 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 que les projets précédents ont servi pour étalonner ces paramètres.
- COCOMO à l'avantage d'être ouvert. Les données de calibrage, les formules et tous les détails des définitions sont disponibles. La participation à son développement est encouragée.
- L'inconvénient : Il 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

2. La méthode COCOMO - suite

3. COCOMO Détaillé ou COCOMO II

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.

2. La méthode COCOMO - suite

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.

2. La méthode COCOMO - suite

3.1. Niveau 1 (Early prototyping model)

- **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.

3.2 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 est prise en compte dans la mesure.

2. La méthode COCOMO - suite

j=1 à 5

4. Paramètres d'estimation

4.1. Modèles Post architecture et Early design

4.1.1. 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: $PM_{NS} = A * (\text{taille})^E * EM$ **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

(**PREC** : L'expérience, **FLEX** :

Flexibilité de développement,

RESL : Architecture/résolution

des risques, TEAM : Cohésion de

l'équipe, PMAT : Maturité du

processus)

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

2. La méthode COCOMO - suite

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 sites distants	1,22	1,09	1,00	0,93	0,86	0,80
SCED	Contrainte sur le délai	1,43	1,14	1,00	1,00	1,00	-

- Le dernier EM_i de la liste (en rouge) n'est pas pris en compte dans l'estimation (SCED)

2. La méthode COCOMO - suite

→ Calcul du délai de développement:

$$C = 3,67 \quad D = 0,28$$

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

3. 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.

3.1 taille en Lignes de code ou en Points de fonction

a) 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.

3.1 taille en Lignes de code ou en Points de fonction

b) 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***)

ANNEXE: 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		

4. Les Points de fonction (FP)

- ✚ proposé par Albrecht en 1979
- ✚ représente le volume de la fonctionnalité d'un système
- ✚ peut être calculé à partir de la spécification du système
- ✚ ajusté en fonction de la complexité du système
- ✚ norme définissant la méthode de calcul est proposée par IFPUG (International Function Point User Group) : <http://www.ifpug.org/>

Domaines d'application

- utilisée principalement pour les systèmes de gestion
- une extension, Full Function Points, a été proposée, en 1999, pour les systèmes
 - temps réel
 - réactifs
 - distribués

4. Les Points de fonction (FP) - suite

Objectifs des points de fonction

- ◆ mesurer la fonctionnalité reçue par l'utilisateur
- ◆ mesurer le développement et la maintenance indépendamment des technologies employées
- ◆ mesure simple applicable à plusieurs notations
- ◆ mesure permettant de comparer plusieurs organisations ou projets différents

Avantages des points de fonction

- basé sur une vue logique du système
- évaluation de la taille d'un progiciel acquis
- évaluation de la taille de la partie utile d'un progiciel (du point de vue de l'utilisateur)
- analyse de qualité et de productivité
- estimation de l'effort de développement
- facteur de normalisation

4. Les Points de fonction (FP) - suite

Types de comptage

- **développement**
 - installation initiale du logiciel
 - comprend la conversion
- **modification et amélioration**
 - ajout, modification ou suppression de fonctions
 - comprend la conversion
- **évaluation de la taille**
 - évaluer la taille d'un système existant

Frontières du comptage

- ✿ **détermine les composantes qui seront incluses dans le comptage**
- ✿ **a une influence sur le type des composantes**
- ✿ **déterminé à partir des objectifs du comptage**

4. Les Points de fonction (FP) - suite

Types de composantes

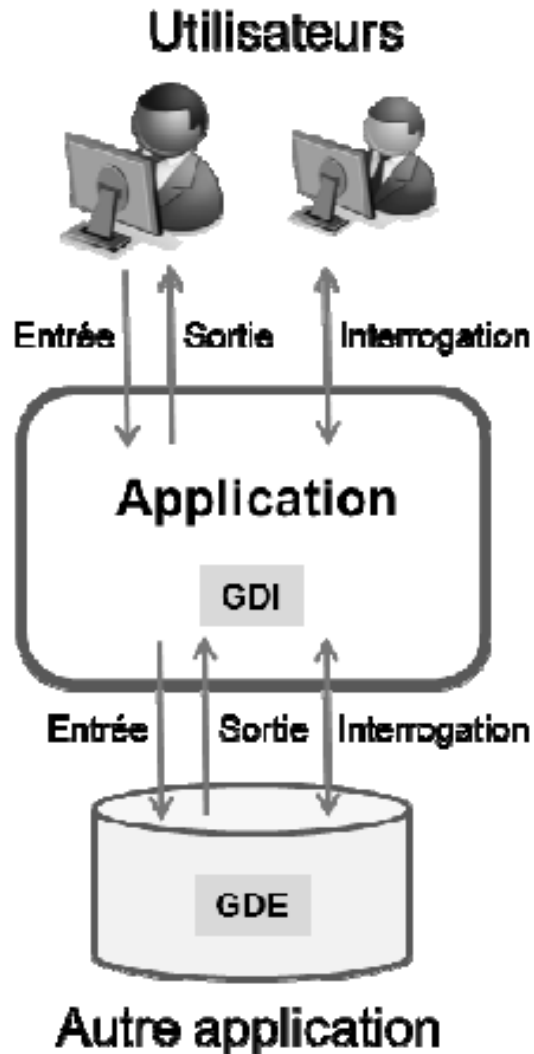
- **Données**

- **groupe logique de données interne (GDI)**
(*ILF - Internal Logical Files*)
- **groupe logique de données externe (GDE)**
(*EIF - External Interface Files*)

- **Transactions**

- **entrée (ENT)**
(*EI - External Inputs*)
- **sortie (SOR)**
(*EO - External Outputs*)
- **interrogation (INT)**
(*EQ - External Inquiries*)

4. Les Points de fonction (FP) - suite



4. Les Points de fonction (FP) - suite

Groupe logique de données interne (GDI)

- à l'intérieur de la frontière
 - groupe de données ou paramètre de traitement
 - identifiable par l'utilisateur
 - mis à jour par le système
- **exemples positifs**
 - entité
 - fichier maître
 - fichiers de trace
 - fichiers de message d'erreur, d'aide
 - tables de code
 - paramètres d'édition ou de contrôle
 - **exemples négatifs**
 - fichiers temporaires, fichiers de sauvegarde, fichiers d'index

4. Les Points de fonction (FP) - suite

Groupe logique de données externe (GDE)

- à l'extérieur de la frontière
- données utilisées par un traitement à l'intérieur de la frontière
- maintenu par un traitement à l'extérieur de la frontière

- **exemples positifs**

- voir GDI

- **exemples négatifs**

- voir GDI

Donnée élémentaire (DE)

- zone non récurrente
- identifiable par l'utilisateur

4. Les Points de fonction (FP) - suite

Sous-ensemble logique de données

- ensemble de données élémentaires
- deux types
 - obligatoires
 - optionnels
- correspond typiquement aux fichiers séquentiels comprenant plusieurs types d'enregistrements

4. Les Points de fonction (FP) - suite

Relation entre GDI/GDE et une base de données

- **il n'y a pas nécessairement une correspondance bijective entre les relations d'une BD et les GDI/GDE**
 - ex: relation pour entête de la facture et relation pour les items de la facture vaut un seul GDI/GDE
 - on peut parfois interpréter les relations formant un GDI/GDE comme des SLD (Sous-ensemble Logique de Données) {SLD: sous groupements de GDI fondés sur la vue logique de l'utilisateur des données}
 - les zones répétées sur plusieurs relations (ex: no de facture) sont comptées une seule fois
- **une relation qui sert exclusivement pour stocker des données temporaires qui ne sont pas fournies par l'usager n'est pas un GDI/GDE**
 - ex: relation qui sert pour bâtir une commande n'est pas comptée
 - ex: relation pour effectuer des calculs d'un processus élémentaire n'est pas comptée

4. Les Points de fonction (FP) - suite

Transactions

- 3 types
 - entrée (ENT)
 - sortie (SOR)
 - interrogation (INT)
- processus élémentaire

1. Processus élémentaire

- plus petite activité qui soit significative pour l'utilisateur
- doit être indépendant
- laisse l'application dans un état de cohérence fonctionnelle
- logique de traitement unique

4. Les Points de fonction (FP) - suite

2. Entrée (ENT)

- données viennent de l'extérieur de la frontière
- fonction de mise à jour d'un GDI
 - ajout. modification et suppression d'enregistrement GDI
 - chaque type compte pour une fonction
 - ex: un écran qui permet l'ajout, la modification et la suppression d'enregistrements compte pour trois fonctions
- **exemples positifs**
 - écran de mise à jour
 - mise à jour en lot
- **exemples négatifs**
 - écran de sélection, de menu, ou de connexion

4. Les Points de fonction (FP) - suite

3. Sortie (SOR)

- génère des données qui franchissent les frontières du comptage
- fait des calculs avec les GDI/GDE
- ne modifie pas les GDI

- **exemples positifs**

- préparation de fichiers de transfert de données entre systèmes
- rapports (calcul sont effectués)
- graphique

4. Les Points de fonction (FP) - suite

4. Interrogation (INT)

- interrogation des données
 - composé d'une entrée et d'une sortie
 - entrée: paramètre de l'interrogation
 - sortie: résultat de l'interrogation
 - ne modifie pas les GDI
 - ne fait pas de calcul
-
- **exemples positifs**
 - écran d'interrogation
 - fonction d'interrogation intégrée à une fonction de mise-à-jour
 - écran d'aide

4. Les Points de fonction (FP) - suite

CALCUL DES POINTS DE FONCTION

$$FP = UFP * VAF$$

- ▶ FP = points de fonction
- ▶ UFP = points de fonction non ajustés
- ▶ VAF = facteur d'ajustement

Calcul des points de fonction non ajustés (UFP)

Calcul des points de fonction non ajustés				
	Complexité			
Type	faible	moyenne	forte	Total
GDI	___ x 7	___ x 10	___ x 15	
GDE	___ x 5	___ x 7	___ x 10	
ENT	___ x 3	___ x 4	___ x 6	
SOR	___ x 4	___ x 5	___ x 7	
INT	___ x 3	___ x 4	___ x 6	
total des points de fonction non ajustés (UFP)				

4. Les Points de fonction (FP) - suite

Détermination de la complexité des données

- basé sur
 - nombre de données élémentaires (DE)
 - nombre de sous-ensembles logiques de données

Détermination de la complexité des transactions

- basé sur
 - nombre de GDI/GDE lus/maintenus (GDR - groupe de données référencés)
 - nombre de DE soumises en entrée, ou produites en sortie, pour faire la transaction

4. Les Points de fonction (FP) - suite

Identification des DE pour une entrée

- zone d'un GDI maintenu par une entrée
 - pas nécessairement fournie par l'utilisateur comme paramètre d'entrée de la transaction; peut-être générée par le système
 - ex: numéro de compte généré automatiquement
- message d'erreur compte pour une seule DE
- l'ensemble des codes d'action et clés de fonction compte pour une seule DE

Identification des DE pour une sortie

- zone apparaissant dans la sortie
- exclure titre, entête, date, heure, constante, etc.

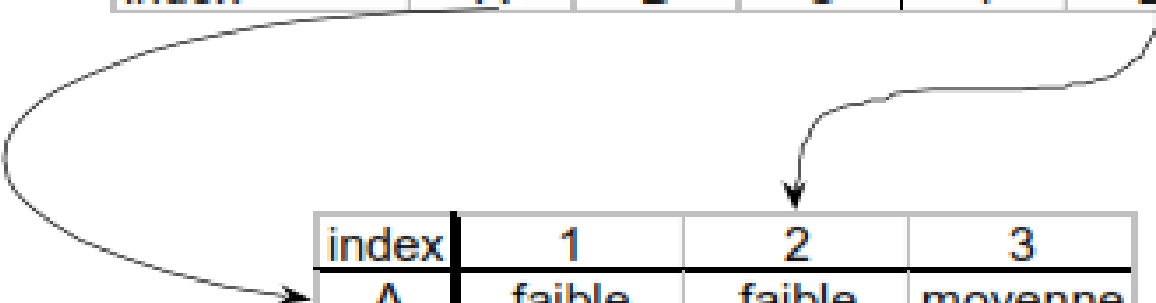
Identification des DE pour une interrogation

- **DE de la partie entrée**
 - zone nécessaire pour sélectionner les données interrogées
- **DE de la partie sortie**
 - zone affichée

4. Les Points de fonction (FP) - suite

Détermination de la complexité

type	nb. GDR / nb. SLD			nb. DE		
GDI	1	2 .. 5	> 5	1 .. 19	20 .. 50	> 50
GDE	1	2 .. 5	> 5	1 .. 19	20 .. 50	> 50
ENT	0 .. 1	2	> 2	1 .. 4	5 .. 15	> 15
SOR	0 .. 1	2 .. 3	> 3	1 .. 5	6 .. 19	> 19
INT	maximum de entrées et sorties					
index	A	B	C	1	2	3



index	1	2	3
A	faible	faible	moyenne
B	faible	moyenne	forte
C	moyenne	forte	forte

4. Les Points de fonction (FP) - suite

Le nombre de points de fonction de chaque composant est obtenu en utilisant la matrice ci-dessous :

	faible	Moyenne	Forte
Entrées	3	4	6
Sorties	4	5	7
Interrogations	3	4	6
GDI	7	10	15
GDE	5	7	10

4. Les Points de fonction (FP) - suite

Exemple de détermination de la complexité

type	nb. GDR / nb. SLD	nb. DE	complexité	poids
GDI 1	1	22	faible	7
GDI 2	6	20	forte	15
GDE 1	3	54	forte	10
GDE 2	2	50	moyenne	7
ENT 1	1	6	faible	3
ENT 2	5	14	forte	6
SOR 1	4	5	moyenne	5
SOR 2	3	20	forte	7
INT 1			moyenne	4
:: entrées	2	6	moyenne	
:: sorties	1	19	faible	
total des points de fonction non ajustés (UFP)				64

4. Les Points de fonction (FP) - suite

Calcul du facteur d'ajustement (VAF)

$$VAF = 0.65 + 0.01 * \sum_{i=1}^{14} F_i$$

F_i : facteur d'ajustement évalué sur une échelle de 0 à 5

F1	télécommunications	F8	m-a-j interactive
F2	traitement distribué	F9	traitement complexe
F3	performance requise	F10	réutilisabilité
F4	configuration saturée	F11	facilité d'installation
F5	taux de transaction	F12	facilité d'exploitation
F6	saisie interactive	F13	plusieurs sites
F7	efficacité utilisateur	F14	configurable

4. Les Points de fonction (FP) - suite

EXEMPLE DE CALCUL DE VAF ET FP

F1	télécommunications	4	F8	m-a-j interactive	2
F2	traitement distribué	3	F9	traitement complexe	5
F3	performance requise	5	F10	réutilisabilité	3
F4	configuration saturée	3	F11	facilité d'installation	5
F5	taux de transaction	5	F12	facilité d'exploitation	2
F6	saisie interactive	1	F13	plusieurs sites	4
F7	efficacité utilisateur	5	F14	configurable	2
				somme des Fi	49

$$VAF = 0.65 + (0.01 * 49) = 1.14$$

$$FP = UFP * VAF = 64 * 1.14 = 73$$

4. Les Points de fonction (FP) - suite

Limitations des points de fonction

- ◆ définition est ambiguë et subjective
- ◆ variance de 10 à 12 % (1 étude) entre estimés produits par différentes personnes
 - variance est moins grande que pour l'estimation des lignes de code
- ◆ comptage est manuel
 - un expert compte environ 57 FP/h-p
 - pour le portfolio de Bell Canada, cela vaut 3 années-personnes

Usage des points de fonction

- on peut bâtir des modèles linéaires de prédiction de l'effort de développement qui sont aussi précis que COCOMO (erreur absolue $\leq 26\%$ dans 81 % des cas)
 - effort = $A + C * FP$
- 1 point de fonction nécessite de 16 à 40 heures-personnes en effort de développement