

Ourouk

Conseil
en Management
de l'Information

La métrologie du logiciel

Tablette n°

Date	2001
Version	V1
Pages	29

Sommaire

1. Objectifs des méthodes de mesure du logiciel	5
1.1. Présentation	5
1.1.1 Métrologie = Développement du logiciel	5
1.1.2 Une partie du coût total du logiciel	5
1.1.3 Evaluer la productivité du développement	5
1.2. Le besoin de méthodologie	5
1.2.1 Une pratique pragmatique	5
1.2.2 Un besoin de rationalité	5
1.3. Des considérations d'un autre ordre	6
1.3.1 Tout n'est pas « objectif »	6
1.3.2 Parkinson et Brooks	6
1.3.3 Aspects mercantiles	6
1.4. Conclusion	6
2. Histoire de la mesure du logiciel (bref aperçu)	8
2.1. Une histoire ancienne	8
2.1.1 Les années 70	8
2.1.2 Une famille de méthodes	8
2.2. Tableau synoptique	8
3. Périmètre de la tablette	10
4. La méthode des points de fonction	11
4.1. Histoire	11
4.1.1 Une traduction mal établie	11
4.1.2 Un standard de fait	11
4.1.3 Des évolutions notables	11
4.1.4 La guerre des méthodes	12
4.2. Exposé de la méthode	12
4.2.1 Une méthode prédictive	12
4.2.2 Les fonctions	12
4.2.3 Cinq catégories	13
4.2.4 Deux parties	13
4.2.5 Complexité des fonctions	13
4.2.6 La pondération des composantes	14
4.2.7 Total	14
4.2.8 Ajustement	14
4.2.9 Coefficient de complexité	15
4.2.10 Total ajusté	16

Sommaire

4.3. Critiques des points de fonction	16
4.3.1 Une méthode peu probante	16
4.3.2 La recherche d'améliorations	16
4.3.3 Deux voies d'amélioration	16
4.3.4 Un champ d'application à délimiter	17
4.3.5 Une mise en œuvre délicate	17
4.3.6 Conclusion	17
5. La méthode COCOMO	19
5.1. Histoire	19
5.2. COCOMO 81	19
5.2.1 Objectif	19
5.2.2 Les modèles	20
5.2.3 Les classes de projet	20
5.2.4 Ajustement	21
5.3. COCOMO II	21
5.3.1 Une redéfinition	21
5.3.2 Grandes évolutions	21
5.3.3 Correspondance	22
5.3.4 Exposant du projet	22
5.3.5 Ajustement	23
5.3.6 Facteurs d'ajustement liés au produit	23
5.3.7 Facteurs d'ajustement liés à la plate-forme de développement	23
5.3.8 Facteurs d'ajustement liés au personnel	23
5.3.9 Facteurs d'ajustement liés au projet	23
5.3.10 Le tableau des valeurs	24
5.3.11 Une géométrie très variable	25
5.4. Critique de COCOMO	25
6. La méthode de Putnam	26
6.1. Objectif	26
6.2. Exposé de la méthode	26
6.2.1 Les travaux de Norden	26
6.2.2 Schéma de la charge	26
6.2.3 Interprétation de la courbe	27
6.2.4 Conséquence	27
6.2.5 Conclusion	27
7. Conclusion	28
7.1. Une démarche qui reste à valider	28

Sommaire

7.2. Un terrain encore à explorer	28
7.3. Des choix plausibles	28

1. Objectifs des méthodes de mesure du logiciel

1.1. Présentation

1.1.1 *Métrologie = Développement du logiciel*

La métrologie est un concept vaste. Dans les cas qui nous intéressent ici, l'objectif est d'évaluer la charge à consacrer au développement logiciel.

1.1.2 *Une partie du coût total du logiciel*

Il s'agit bien du développement et non des coûts qui peuvent lui être associé (vente, commercialisation, maintenance, formation, etc.) et qui forment ce qui est souvent appelé le coût total de possession (TCO, Total cost of ownership). Ces derniers coûts se révèlent d'ailleurs souvent plus importants que le développement proprement dit.

1.1.3 *Evaluer la productivité du développement*

Il s'agit, par la même occasion, d'évaluer la productivité du développement. Ce dernier point doit être manié avec prudence car il peut provoquer des réactions négatives des développeurs et, par la même occasion, ruiner les avantages que l'on pourrait tirer de la mesure.

1.2. Le besoin de méthodologie

1.2.1 *Une pratique pragmatique*

Dans les entreprises, on est amené, notamment lors de la vente, à estimer et prédire des coûts. Ces estimations reposent à la fois sur des données objectives liées à l'expérience et la connaissance des acteurs. Les professionnels, par analogie avec des expériences passées et de par leur connaissance des processus à mettre en œuvre et des temps relatifs qui y sont associés, sont à même d'évaluer la charge associée au développement logiciel. Même si, dans de nombreux projets, il existe des écarts importants entre les estimations et la réalisation, on doit constater que, dans l'ensemble, ces professionnels parviennent bien à prédire les coûts puisque leurs sociétés continuent d'exister.

1.2.2 *Un besoin de rationalité*

Ce constat ne doit pas pour autant nous laisser nous abandonner à une quiétude irraisonnée. En effet, il existe des projets dont l'importance peut être critique pour une société et pour lesquels l'erreur d'évaluation est catastrophique. D'autre part, il est important de disposer de méthodes qui ne sont pas liées uniquement au savoir-faire des individus. Leur existence garantit une plus grande objectivité de la mesure, une plus rapide capacité d'évaluation, une facilitation de la transmission des procédures et des démarches entre les générations au sein de l'entreprise. L'utilisation de ces méthodes permet aussi à l'entreprise de réfléchir plus facilement sur l'analyse des écarts par rapport à la prévision. Last but not least, du point de vue de la démarche commerciale, elle présente aussi pour le client l'avantage de l'objectivité de l'évaluation et la maîtrise des coûts de développement.

1.3. Des considérations d'un autre ordre

1.3.1 *Tout n'est pas « objectif »*

Tout entrepreneur sait aussi que la prédiction du coût du développement ne prend pas seulement en compte le jugement « objectif » des professionnels qui se basent sur leur savoir-faire mais aussi des composantes organisationnelles et mercantiles irréductibles.

1.3.2 *Parkinson et Brooks*

Les composantes organisationnelles sont en général mises en relief par deux caricatures qui, tout en forçant le trait, montrent bien l'importance de l'organisation.

Il s'agit de la loi de Parkinson et de la loi de Brooks.

Parkinson était un professeur britannique. Il écrivit à la suite de son engagement dans l'Etat major de l'armée britannique durant la seconde guerre mondiale, un petit chef d'œuvre d'humour et de management (1=2 ou les règles d'or de Mr Parkinson).

Il y exprime notamment ce qui est devenu la « loi de Parkinson »¹ et qui peut se résumer ainsi : « Le travail s'étend jusqu'à remplir tout le temps alloué à son exécution ». En illustration de cette « loi », Parkinson montrait que le nombre d'employés du Ministère des colonies n'avait jamais été aussi important qu'au moment où la Grande-Bretagne n'avait pratiquement plus de colonies.

La « loi de Brooks », (de Frédéric Brooks, Directeur de projet chez IBM et auteur d'un livre critique sur l'évaluation en homme x mois des projets²), est plus pragmatique « Ajouter du personnel à un projet en retard ne fait que le retarder ».

1.3.3 *Aspects mercantiles*

Les composantes mercantiles sont souvent prépondérantes dans l'évaluation d'un projet. Partant du fait qu'une entreprise doit, avant de devoir réaliser et même de pouvoir réaliser, vendre, les logiques propres à la demande peuvent s'imposer. De ce point de vue les questions que se pose l'entreprise pour évaluer son prix ressemblent aux questions suivantes :

Quel prix le client est-il prêt à accepter ? De quel budget dispose t-il ? Quel prix les concurrents vont proposer ? Nous n'avons pas de référence sur le projet que nous demande le client. Devons nous l'obtenir coûte que coûte ? Une fois dans la place pourrions nous obtenir des travaux supplémentaires ou de nouvelles offres ? etc.

1.4. Conclusion

C'est donc une synthèse de tous ces aspects qui nourrit la décision. Les critères qui justifient l'évaluation du projet ne sont pas

¹ Parmi les autres « lois » de Parkinson on trouve :

- Les sujets les plus débattus dans une réunion sont les sujets mineurs car tout le monde en comprend l'importance (corollaire : Plus les sommes en jeu sont importantes, moins elles sont correctement appréciées.)
- L'efficacité d'une réunion décroît dès qu'elle passe de 5 à 8 personnes. Etc.

² Dans son ouvrage intitulé le mythe de l'homme x mois, Brookes fait notamment remarquer que la notion d'homme x mois n'est pas totalement commutative. On ne peut pas nécessairement transformer les hommes en mois et inversement. En effet, plus il y a de personnes sur un projet plus on perd de temps à coordonner les acteurs.

Les critères qui justifient l'évaluation du projet ne sont pas nécessairement identiques d'un projet à l'autre, d'un client à l'autre. Les méthodes de mesure du logiciel tentent de fournir du point de vue de l'offre, une démarche rationnelle d'évaluation et de prédiction des coûts.

2. Histoire de la mesure du logiciel (bref aperçu)

2.1. Une histoire ancienne

La nécessité de mesurer la productivité du développement informatique est aussi ancienne que cette activité. Beaucoup de méthodes ont été proposées pour répondre à cette question. Aucune n'a emporté une adhésion définitive ni ne s'est montrée probante. Certaines méthodes ont cependant remporté plus de succès que d'autres et d'une certaine manière c'est aujourd'hui la méthode par les points de fonction qui obtient la plus forte notoriété suivie par la méthode COCOMO.

2.1.1 Les années 70

On doit constater que la plupart des méthodes qui sont encore citées aujourd'hui sont nées à la fin des années 70. Elles font suite aux premières méthodes qui évaluaient la charge de travail en fonction du nombre de lignes de programme. Les difficultés rencontrées dans ce type d'évaluation comme le développement de la psychologie cognitive ont ouvert une nouvelle perspective. Avec Halstead commence la recherche de l'évaluation de l'effort intellectuel attaché au développement logiciel, d'où l'emploi fréquent du terme effort comme synonyme de charge de travail. La méthode des points de fonction se situe dans cette tradition sans faire appel comme Halstead à des travaux douteux de psychologie cognitive.

2.1.2 Une famille de méthodes

Plus qu'à une méthode nous avons aussi à faire à une famille de méthodes. Du moins pour les deux dont on peut dire qu'elles se sont maintenues, à savoir les points de fonction et COCOMO.

De nombreux développements parallèles ont eu lieu pour les « améliorer ». Il s'y ajoute les tentatives de constituer des méthodes privées qui tout en reprenant les bases des méthodes existantes procèdent à des adaptations et des raffinements susceptibles de créer une différenciation. L'Europe via de nombreux projets a tenté de combler ce qui pouvait se présenter comme un retard vis-à-vis du monde anglo-saxon mais n'est jamais parvenue à les faire connaître au delà d'un cercle restreint d'experts.

2.2. Tableau synoptique

Date	Auteur	Titre	Brève description
1975	Halstead	Software science. Théorie de l'information. Psychologie cognitive	Le temps de développement est fonction du nombre d'opérateurs et du nombre d'opérandes dans le vocabulaire du programme
1976	Mc Cabe	Complexité cyclomatique Théorie des graphes	Cette méthode mesure la complexité du programme (nombre cyclomatique)

1978	L Putnam		Le modèle de Putnam vise surtout à optimiser la répartition de la charge dans un projet.
1979	Allan Albrecht	Points de fonction	Estimation de l'effort à partir de l'estimation de la taille de l'application
1981	Barry Boehm	Cocomo	
1984	Allan Albrecht	Points de fonction révisée	
1987	Caper Jones	Features Points	
1988	Charles Symons	Mark II	Evolution des points de fonction
1994	IFPUG	Points de fonction révisée 4.0	
1997	Barry Boehm	Cocomo II	Refonte de la méthode Cocomo
1997	Alain Abran	FFP	Points de fonction étendus
2000	Alain Abran Charles Symons Etc.	COSMIC-FFP	Convergence des méthode par points de fonction

3. Périmètre de la tablette

Au sein de cette masse de méthodes nous avons choisi de développer celles qui :

1. ont pour objectif d'évaluer et prédire la charge
2. ont une notoriété importante
3. ont les meilleurs résultats comparatifs
4. sont publiques

Par conséquent, nous verrons plus particulièrement la méthode des points par fonction et le modèle COCOMO.

4. La méthode des points de fonction

4.1. Histoire

4.1.1 Une traduction mal établie

La traduction la plus fréquente de la méthode est points de fonctions (function points) bien que la traduction points par fonction nous paraisse plus explicite de son contenu.

4.1.2 Un standard de fait

Il s'agit d'une des méthodes les plus populaires, sinon la plus populaire. Elle est devenue la norme de fait dans le domaine de l'informatique de gestion.

La méthode a été créée par Allan Albrecht, d'IBM.

Elle a été publiée pour la première fois en 1979 lors d'une conférence IBM.

Une première révision fournissant une version plus élaborée a été publiée en 1983 dans une revue arbitrée.

A peu près à la même époque, un groupe d'utilisateurs de la méthode a été créé (IFPUG - International Function Points User Group).

Ce groupe s'est donné pour but le développement de la méthode comme sa clarification et sa normalisation³.

De 1987 à 1994, 4 révisions des règles et des normes ont été effectuées.

La version 1994 de la méthode a été publiée par l'AFNOR, en Avril 1995, sous forme de norme expérimentale (XP Z 67-160).

Le fait que cette norme existe encore sous sa forme expérimentale 6 ans⁴ après montre qu'elle n'a guère rencontré de public. Les groupes d'expert à l'origine de cette norme ont cessé de fait leur activité. L'AFNOR est sur ces sujets beaucoup plus impliquée dans les normes qualité que dans les normes de mesure.

L'IFPUG en est depuis peu à sa version 4.1.

4.1.3 Des évolutions notables

Les difficultés rencontrées par la méthode dans le domaine des logiciels en temps réels, comme celles résultant de tentatives d'automatisation du calcul des points de fonction à partir de l'analyse des programmes⁵ ont poussé les équipes canadiennes dirigées par le professeur Alain Abran à mettre au point une méthode dite « points de fonction étendus ».

Cet effort de création d'une deuxième génération de méthodes de

³ En France, le FFUP (French Function Point User's Group) créé en 1992 a la même vocation. Il semble avoir disparu.

⁴ Dans la règle une norme expérimentale est revue fait l'objet d'un bilan au bout de trois ans.

⁵ « ...nous avons constaté:

- que les travaux à date des autres chercheurs à travers le monde n'avaient produit que des approximations, et qu'il était même impossible de préciser ces approximations;
- qu'il y avait des problèmes structuraux majeurs à la méthode actuelle des 'points de fonction', ce qui en rendait impossible une automatisation avec un niveau de précision raisonnable.. » Alain Abran (correspondance privée)

mesure fonctionnelle de la taille des logiciels est relayé par un groupe d'expert internationaux (COSMIC Common Software Measurement International Consortium) fondé en 1998 et promoteur de la méthode COSMIC-FPP (FFP = Full Function Points) ou COSMIC-PFE soit Points de fonction étendus.

D'une certaine manière la méthode COSMIC se veut être au confluent et réunifier de nombreuses méthodes qui s'étaient développées à partir de la méthode d'Albrecht⁶ pour tenir compte de la spécificité des logiciels qui échappent au domaine des logiciels de gestion soit 10 à 20% des logiciels de l'industrie électronique et des télécommunications.

La méthode COSMIC se pense donc comme le prolongement, l'extension et le raffinement des méthodes FFP, MarkII, IFPUG 4.1, tout en se situant dans le cadre des normes ISO propres à la mesure du logiciel (ISO 14143).

4.1.4 La guerre des méthodes

Mais la guerre des méthodes fait rage. Arguant qu'elle était la seule méthode pour l'industrie, l'IFPUG a demandé une procédure accélérée de normalisation à l'ISO.

L'IFPUG a perdu beaucoup de son influence et est devenu un organisme dans lequel les américains jouent un rôle majeur. Cette démarche a entraîné des réactions britanniques. Ils ont demandé pour la méthode Mark II une procédure accélérée pour devenir une norme ISO (iec/dis 20968). C'est également le cas des néerlandais (méthode NESMA) qui ont procédé à la même démarche.

Comme par ailleurs, la méthode COSMIC-FFP fait également l'objet d'un processus de normalisation ISO et que des contestations s'élèvent sur la démarche IFPUG, la procédure accélérée risque de se ralentir...

4.2. Exposé de la méthode

4.2.1 Une méthode prédictive

Cette méthode a pour but de fournir une méthode prédictive, disponible avant le développement et indépendante de la technologie. Elle repose donc sur la décomposition technique initiale des fonctionnalités du logiciel. Plus l'estimation s'appuiera sur les fonctionnalités décrites dans les spécifications détaillées plus sa précision sera forte. Mais dans la règle c'est bien avant que doit se faire l'estimation. Il s'ensuit une forte probabilité d'erreur car il est fréquent qu'un écart important existe entre « l'idée » du logiciel et le logiciel « réel ».

4.2.2 Les fonctions

La première étape du calcul des points par fonction consiste à établir la liste, la complexité et le nombre des fonctions. Nous avons déjà vu que la méthode de mesure par les points de fonction avait connu une première évolution entre 1979 et 1984 puis de nouvelles précisions dans le cadre de l'IFPUG. Pour cet exposé nous tiendrons compte

⁶ Certaines ne sont plus utilisées (Feature points - 1987, 3-D FP's), d'autres n'interviennent que dans le champ d'une société (Boeing 3D -1994). En revanche la méthode Mark II (MkII) a conquis plus de la moitié du marché britannique, en est dans sa version 1.3.

sur la base de l'analyse de 1984 réalisée par le fondateur de la méthode (Albrecht) des éventuelles conceptualisations qui ont suivi⁷ (Symons fondateur du Mk II, IFPUG version 4)

Par rapport au modèle de 1979, le modèle de 1984 introduit, notamment, une plus grande complexité dans le calcul des points.

4.2.3 Cinq catégories

Le processus de traitement de l'information est décomposé en cinq grandes composantes :

- Les entrées. Les entrées sont notamment les formulaires via lesquels un utilisateur peut introduire des données dans l'application.
- Les fichiers logiques internes, devenus dans la terminologie IFPUG, les groupes de données internes. On peut les assimiler aux entités dans les modèles de données.
- Les fichiers logiques externes, devenus dans la terminologie IFPUG, les groupes de données externes. Les fichiers externes sont les fichiers partagés appartenant à d'autres applications.
- Les interrogations. Les interrogations ou questions de l'utilisateur se traduisent par des dialogues homme machine (consultation d'un menu contextuel, d'une aide, requêtes, ...)
- Les sorties. Les sorties seront les vues et les états fournis par l'application à l'utilisateur.

4.2.4 Deux parties

Dans ces cinq catégories on a tendance à distinguer deux parties en relation avec le processus de traitement de l'information :

- les données qui sont conservées, stockées
- les transactions qui représentent l'aspect dynamique du traitement de l'information

Dans les données on fait figurer les points de 2 et 3 soit les groupes de données internes et externes.

Les transactions sont constituées par les autres composantes : les entrées, les sorties, les interrogations.

4.2.5 Complexité des fonctions

Pour chaque grande fonction, il s'agit de déterminer le nombre de composantes et sa complexité. La complexité varie selon une échelle pré-définie (basse, moyenne, haute).

Par exemple, pour les groupes de données internes on peut s'appuyer sur le modèle conceptuel des données. Dans un modèle entité relation on détermine les entités primaires. Ce sont typiquement les objets des bases de données - tables primaires -. On évalue ensuite le nombre de relations entre les tables. Au sein des tables on compte le nombre d'éléments. Ces éléments permettent d'attribuer un niveau de complexité⁸.

Prenons, par exemple, un modèle de données simple où une table 1

⁷ La conception d'Albrecht reste tributaire d'un environnement IBM.

⁸ Pour les groupes de données internes le tableau est le suivant :

	1-19 éléments	20-50 éléments	51 et + éléments
1 relation	Basse	Basse	Moyenne
2 à 5 relations	Basse	Moyenne	Haute
6 relations et plus	Moyenne	Haute	Haute

est associée à deux autres par une relation 1 à plusieurs. Supposons que la table 1 comporte 15 champs contre 6 champs pour la table 2 et 4 pour la table 3.

Nous aurions 3 entités (les tables de paramètres ne sont pas prises en compte). L'entité Table 1 a deux relations et moins de 20 éléments. Sa note de complexité est donc basse. C'est également le cas pour les deux autres entités (1 seule relation et moins de 20 éléments). Dans cet exemple, nous avons donc 3 groupes de données internes de complexité basse.

La même chose vaut pour les autres fonctions en tenant compte de leurs caractéristiques.

Les groupes de données externes sont utilisés par l'application mais ne sont pas maintenus par elle.

Les entrées se caractérisent par le fait qu'elles apportent des changements dans les données se trouvant dans l'application. Ces modifications peuvent résulter d'une entrée directe comme d'une mise à jour par lot. L'important est qu'il y ait une modification des données que ce soit par création, modification ou suppression (chacune de ces fonctionnalités est comptée comme un entrée).

Les sorties sont les résultats de l'application. C'est le cas des Etats, des Vues. Comme pour les entrées on tient compte du nombre d'éléments et du nombre d'entités qui sont sollicités dans les groupes de données.

Les interrogations ou les questions supposent un couple requête/réponse. Il s'agit d'une réponse directe à une requête mais pas nécessairement d'une réponse immédiate.

4.2.6 La pondération des composantes

Une fois l'inventaire des composantes réalisé, on multiplie les éléments du tableau des composantes par les éléments de pondération des composantes suivant leur complexité. Ce tableau est défini a priori par la méthode à partir des retours d'expérience.

Le tableau de pondération des composantes selon leur complexité est le suivant :

Composantes	Complexité basse	Complexité moyenne	Complexité haute
Entités internes	7	10	15
Entités externes	5	7	10
Entrées	3	4	6
Sorties	4	5	7
Interrogations	3	4	6

4.2.7 Total

En faisant la somme des 15 produits on obtient le total des points de fonction bruts⁹ (UFP unadjusted fonction points).

4.2.8 Ajustement

Ce total des points de fonction bruts peut être ajusté en fonction de divers critères liées aux caractéristiques attendues par l'utilisateur. Le questionnaire qui sert à ajuster les points comporte 14 facteurs qui

⁹ Dans notre exemple, les entité internes auraient obtenu 21 points de fonction. 3 entités de complexité basse x 7 donnent 21 points de fonction.

font l'objet de réponses à choix multiples. Suivant la réponse qui est cochée on attribue une note qui varie de 0 à 5.

Ces facteurs sont les suivants :

1 Transmission des données (application traitée par lot -batch-, entrée des données à distance, télétraitement interactif.

2 Distribution. On évalue ici la contribution de l'application au transfert des données (préparation des données sur une autre système, traitement dynamique des données sur une élément du système, etc.).

3 Performance. On évalue ici, le fait se savoir si des objectifs de performance ont été dictés ou non par l'utilisateur.

4 Configuration. On mesure ici l'existence éventuelle de restrictions concernant l'exploitation de l'application.

5 Taux de transaction. Plus le taux de transactions est élevé plus il a d'influence sur le conception, la mise en œuvre et le support de l'application.

6 Entrées des données en temps réel. Ce point du formulaire évalue l'importance des fonctions de commande et d'entrées de données interactives.

7 Convivialité. Ce point permet dévaluer les contraintes éventuelles posées par l'utilisateur concernant la convivialité et l'efficacité de l'application pour de dernier.

8 Mise à jour en temps réel. On évalue ici si les entités internes de l'application sont mises à jour en ligne. On tient compte aussi de la protection contre la perte des données et du coût de la reprise.

9 Complexité du traitement. On prend en compte la complexité du traitement défini de diverses manières : importance de la sécurité et du contrôle, importance du traitement logique et mathématique, importance des exceptions dans le traitement, importance des transactions devant être retraitées, etc.

10 Réutilisation. On mesure ici l'effort accompli pour faciliter la réutilisation des programmes dans d'autres applications.

11 Facilité d'implantation. On évalue ici la facilité d'installation et notamment l'importance des conversions et leur degré de planification et de maîtrise.

12 Simplicité d'utilisation. Cette rubrique évalue la facilité d'exploitation de l'application. Procédures de démarrage, de sauvegarde, de relance, manipulation de papier, etc.

13 Installations multiples, portage. On évalue ici le fait de savoir si l'application a été conçue pour être mise ne place dans des installations multiples.

14 Facilité des modifications, maintenance. On mesure ici, le fait de savoir si l'application a été conçue pour faciliter les modifications (fonctions d'interrogations souples, etc.)

4.2.9 Coefficient de complexité

Ces 14 facteurs vont nous permettre de déterminer un coefficient de complexité du projet. On fait la somme des réponses (valeur comprise entre 0 et 5 pour chaque réponse) fournies à l'ensemble des facteurs recensés dans le questionnaire. On peut donc obtenir

une somme variant de 0 à 70. La moyenne théorique est donc de 35. Lorsque cet indicateur de complexité est conforme à la moyenne, il n'y a pas de différences entre le total des points de fonction bruts et le total ajusté. Par conséquent le coefficient d'envergure du projet, le facteur de complexité technique du projet, est égal à $0,65 + 0,01 \times$ (Somme des 14 facteurs). La somme des 14 facteurs, nous l'avons vu est comprise entre 0 et 70.

4.2.10 Total ajusté

Pour obtenir le nombre total des points de fonction on multiplie le nombre de points de fonction bruts par le facteur de complexité technique.

Points de fonction ajustés = Points de fonction bruts x facteur de complexité technique. (PFA = PFB x FCT ou selon les sigles anglo-saxons FP = UFP x TCF)

A supposer maintenant que l'on connaisse le nombre de jour x homme par point de fonction (il résulte de l'expérience), on peut évaluer la charge potentielle.

Charge de travail pour le développement (en jour x homme = Points de fonction ajustés x nombre de jour x homme par point de fonction)

4.3. Critiques des points de fonction

4.3.1 Une méthode peu probante

En premier lieu cette méthode ne s'est jamais montré probante. Toutes les mesures qui l'ont testé sur une masse de projets pour lesquels on peut considérer que l'information obtenue était cohérente et la méthode de comptage homogène montrent que le facteur explicatif des points de fonction avoisine les 50% et n'est guère amélioré par l'ajustement. Dans un autre contexte, selon le témoignage d'un Directeur d'une grande SSII française, sur plus de 2000 projets les écarts varient de 1 à 30 entre la mesure et la réalité.

D'autre part, comme nous l'avons déjà noté dans l'introduction, du jour où l'estimation est faite sur la base du cahier des charges fonctionnel, le risque d'une divergence avec le logiciel réel n'est pas négligeable. Les mesures de ce genre ont montré une erreur de 1 à 5 entre le projet initial estimé et le projet réalisé.

4.3.2 La recherche d'améliorations

Très tôt, les critiques des expérimentateurs se sont manifestées. C'est le cas par exemple de Charles Symons qui avait utilisé la méthode quand il était chez Xerox. Mais le concept a moins reçu de remises en cause que de volonté de l'améliorer. Suivant l'idée qu'on se faisait des origines du décalage estimation/réel on a plutôt cherché à perfectionner le modèle. Cela explique la naissance de nombreuses méthodes cherchant à adapter et perfectionner la méthode initiée par Albrecht. Entre 1979 et 1984, Albrecht a d'ailleurs perfectionné la méthode, jugée trop sommaire par certains critiques.

La démarche des points ajustés introduit donc un potentiel de flexibilité non négligeable puisque à partir d'un périmètre fonctionnel brut identique on peut faire varier pratiquement du simple au double (1,35/0,65) le nombre de points final.

4.3.3 Deux voies

Deux grandes tendances se sont dégagées dans l'application des

d'amélioration

règles de base.

Une tendance a cherché à se baser sur le type de base de données de l'entreprise (IBM, par exemple, dont Albrecht fait partie, prenant comme point de départ la base de données IMS) tandis que l'autre cherchait à bâtir la mesure des composantes sur les modèles logiques des entreprises.

Cette dernière tendance a notamment été représentée par Charles Symons qui, en 1988, proposait de reprendre le modèle d'Albrecht en y ajoutant un facteur d'environnement (Personnel, organisation, outils et techniques, environnements physiques, standards, etc.) pour prendre en compte une complexité jugée insuffisamment traitée par le modèle. Cette démarche rejetée par l'IFPUG est à la base de la méthode Mark II.

4.3.4 Un champ d'application à délimiter

Outre d'être tributaire d'un environnement IBM pour l'élaboration des tableaux de pondération, on a reproché à la méthode des points de fonction d'introduire des biais dans l'évaluation de projets scientifiques car ce type de projet n'était pas représenté dans l'échantillon d'Albrecht.

Comme les projets scientifiques ont beaucoup plus de traitement internes et de calculs complexes et donc moins de transactions et de relations avec l'utilisateur les points de fonctions peuvent être sous-estimés. Il en va de même pour les projets du domaine militaire où les contrôles sont plus rigoureux.

C'est à ces questions que tentait de répondre la méthode « feature points » développée par Capers Jones. La nécessité de mieux tenir compte des environnements de calcul dans notamment les systèmes en temps réel ont conduit, comme nous l'avons vu, vers la mise au point de la méthode des points de fonction étendus.

4.3.5 Une mise en œuvre délicate

Outre les critiques méthodologiques, la méthode des points de fonction a fait l'objet de critiques pratiques. Elles portent sur la difficulté de mise en œuvre. D'une part, il y a les difficultés liées à l'évaluation des points de fonction. Il n'est pas toujours aisé de définir les frontières et les limites d'une application ni de la matière à compter. Les guides et les règles d'utilisation essaient de baliser le mieux possible les démarches mais on ne peut exclure des difficultés concrètes d'interprétation et donc de cohérence des mesures (on estime à 10% la différence d'appréciation d'une personne à l'autre). Par ailleurs, l'utilisation de la méthode peut demander un travail important et un accompagnement par un spécialiste. L'automatisation du comptage ne peut pas encore actuellement être réalisée¹⁰. D'autre part, il reste à partir des projets (un référentiel important est à bâtir) à établir la valeur du point de fonction.

4.3.6 Conclusion

Bien qu'il y ait des défenseurs d'autres méthodes, nous allons le voir avec COCOMO de Barry Boehm, le concept même de la méthode des points par fonction a plus rarement été remis en cause. On a plutôt cherché à perfectionner la méthode ou mis l'accent sur son champ d'application ou sur les difficultés pratiques mais explicitement ou implicitement on a souscrit à l'idée qu'il y avait une relation entre

¹⁰ En tout état de cause, cette automatisation ne pourrait être envisagée que dans le cadre de la nouvelle méthode COSMIC-FFP

l'importance des fonctionnalités et l'effort de développement.

Des lois générales relatives à l'évolution de la productivité du travail, on doit déduire que pour un périmètre fonctionnel donné, la charge correspondante doit baisser. Par exemple, les langages de programmation sont plus productifs et permettent de réaliser en moins de temps un développement. Au passage on notera que cette productivité accrue ne s'applique pas uniformément le long de la chaîne de production logicielle. Le temps consacré à l'écriture des programmes diminue relativement (il est passé de la moitié du temps de développement au 1/3 environ, tandis que les temps de test et d'installation progressent¹¹.

Mais le constat d'une baisse du temps moyen par point de fonction n'est pas une évidence. Pour prendre l'exemple de l'automobile, le prix en euros constant des automobiles ne varie guère. Si le périmètre fonctionnel se définissait par le fait qu'il y ait, des portes pour entrer, un tableau de bord, un volant et des pédales pour passer les commandes, des sièges pour s'asseoir, un coffre pour les bagages, un moteur plus ou moins complexe en fonction de la cylindrée et du nombre de cylindres, on peut dire que ce périmètre fonctionnel n'a guère varié et comme le véhicule demande toujours une charge de travail comparable on en déduirait une certaine stabilité de la charge par point de fonction. Ce serait passer à côté de nombreuses améliorations qui ne sont pas ou mal mesurées par le périmètre initial (la sécurité, la qualité, la finition, l'esthétique, la moindre pollution, la moindre consommation, la convivialité, l'assistance électronique pour le freinage, la direction, la stabilité du véhicule, la climatisation, etc.). il en va de même pour le logiciel. Un utilisateur sera sensible à la qualité d'un système, à sa facilité à traiter les données, aux performances de l'exploitation, à sa convivialité, ses évolutions esthétiques. Les modèles ajustés s'efforcent en partie de prendre en compte mais plus fondamentalement se pose la question du périmètre fonctionnel. Or ce la définition de ce périmètre est beaucoup plus complexe qu'il n'y paraît ; il s'évanouit ou est repoussé plus loin quand on croît l'atteindre. Cela rend d'autant plus difficile les comparaisons et a fortiori les comparaisons dans le temps (par exemple, les interfaces graphiques se sont développées, l'usage de la souris généralisé, etc.).

¹¹ En 1975, Brooks prétendait avoir un certain succès avec la « règle du pouce » (1/3 planification, 1/6 programmation, 1/4 tests des composants et premier test du système, 1/4 test du système)

5. La méthode COCOMO

5.1. Histoire

Revenant sur les pas des méthodes les plus anciennes, Barry Boehm¹² a développé la méthode COCOMO (COConstructive COSt Model) dans son livre *Software Engineering Economics* paru en 1981.

Des ajouts sont apportés en 1987 avec Ada COCOMO qui recevra par la suite quelques modifications. La méthode reflétait bien (à l'instar des autres) l'état de la technologie dans ces années là. L'évolution rapide de l'informatique périssait, de l'aveu même de ses auteurs, bien des considérations du modèle COCOMO et le rendait toujours moins opérationnel. Par exemple, il avait été créé à une époque où le traitement par lot (batch) était monnaie courante et un temps d'attente était nécessaire au programmeur avant de recevoir le résultat de son travail alors qu'aujourd'hui le temps réel, l'interactivité, les utilisateurs multiples sont l'environnement traditionnel du développement logiciel.

D'autre part, la perspective de la réutilisation des programmes comme le développement de composants, la mise en place de méthode de développement rapide, l'utilisation de progiciels et de langages évolués modifiant considérablement le cadre d'action de COCOMO.

De nouveaux travaux sont engagés en 1994. A partir de 1995 apparaissent les premiers résultats décrivant la méthode COCOMO 2.0. En 1997, une première version nommée COCOMO II. 1997 paraît. Une deuxième version est produite en 1998 qui sera reconduite jusqu'en 2000, date où paraît le nouveau livre de Barry Boehm "*Software Cost Estimation with COCOMO II*". A partir de 1997 on prend l'habitude de nommer COCOMO 81, la première méthode parue en 1981 et COCOMO II la méthode initiée à partir de 1994 et qui connaît des améliorations successives depuis cette date.

5.2. COCOMO 81

5.2.1 Objectif

COCOMO cherche à mesurer l'effort à partir de la taille du programme source. La méthode se raffine ensuite en introduisant divers facteurs correctifs.

L'équation d'estimation de la charge est du type

¹² Mathématicien de formation, il travaille chez Rand (1959-1973) où il est conduit à mettre en évidence que les échecs d'un système de commande et de contrôle sont dus au logiciel. Il renoue donc avec des préoccupations qu'il avait eu comme stagiaire chez General Electric sur les difficultés de la construction d'un logiciel. Il travaille ensuite dans une société d'ingénierie et de production tournée vers l'aérospatiale (TRW) (1973-1989). Il se dédie désormais à l'enseignement et la recherche en Californie où il dirige notamment le centre d'études du logiciel à l'université de Californie du sud, qui veille à la mise à jour des différentes méthodologies dont il est le père. Outre COCOMO, il est l'auteur du modèle en spirale du développement logiciel et de la théorie W (win-win) pour le management des projets logiciels.

Charge totale (estimée en hommes x mois) = a tailleⁿ m(X)

Où :

- a et n sont des paramètres fournis par la méthode COCOMO en fonction du mode de travail lors du projet logiciel et des modèles COCOMO. Boehm a lui même extrapolé ces paramètres de l'analyse de nombreux projets comportant un nombre d'instructions source variant de 2000 à 1 000 000.

- m(X) est un facteur d'ajustement de la charge.

- la taille est exprimée en KDSI (Kilo Deliverable Source Instructions), milliers d'instructions sources délivrables, hors commentaires. Du point de vue de la prédiction du coût nous sommes dans la même situation que dans le cas d'une évaluation en hommes x mois. En effet, pour déterminer la charge, il faut évaluer la taille du logiciel. Mais il semblerait qu'un professionnel soit plus à l'aise pour évaluer une taille dans un langage donné.

5.2.2 Les modèles

COCOMO 81 distingue trois modèles : simple, intermédiaire et détaillé.

- Le modèle simple à un facteur d'ajustement de 1.
- Le modèle intermédiaire ajoute notamment au modèle simple un facteur d'ajustement de la charge variable.
- Le modèle détaillé ajoute à l'estimation globale du système sur la base du modèle intermédiaire une estimation séparée des sous systèmes.

5.2.3 Les classes de projet

COCOMO 81 distingue également trois classes de projets : organique, semi-détaché, intégré

- Un projet organique rassemble de petites équipes qui travaillent sur les applications de l'entreprise. Elles ont une bonne connaissance de l'environnement matériel et logiciel comme du contexte. Peu de communications sont nécessaires.
- Un projet semi-détaché rassemble des équipes qui ont une expérience limités du type d'application qui est développée ainsi que du système utilisé.
- Un projet intégré suppose un environnement matériel et logiciel complexe. Il existe des contraintes particulières comme la complexité des calculs, des contrôles ou de la sécurité. Les besoins ne peuvent pas être modifiés pour pallier des difficultés logicielles.

Ces différences se traduisent par des coefficients particuliers pour évaluer la charge.

COCOMO	Organique	Semi-détaché	Intégré
COCOMO simple			
a	2,4	3	3,6
n	1,05	1,12	1,2
COCOMO intermédiaire			
a	3,2	3	2,8

n	1,05	1,12	1,2
---	------	------	-----

5.2.4 Ajustement

A ces facteurs, il faut ajouter, pour le modèle intermédiaire, le facteur d'ajustement de charge qui dépend de 15 paramètres (fiabilité, taille de la base de données, complexité du produit, contrainte de temps d'exécution, contrainte de taille mémoire, instabilité de la machine virtuelle, temps de retour des travaux, compétence des analystes, compétence des programmeurs, expérience du type d'application, expérience de la machine virtuelle, expérience du langage de programmation, utilisation d'outils logiciels, utilisation des méthodes de programmation, contrainte de délai).

Ces paramètres sont classés sur une échelle de valeur à 6 positions (Très basse, basse, nominale, haute, très haute, extra haute). Des points dont l'amplitude peut varier de 0,7 à 1,66 sont affectés à ces positions.

5.3. COCOMO II

5.3.1 Une redéfinition

Pour tenir compte des évolutions dont nous avons parlé plus haut, le modèle COCOMO a été redéfini.

D'un modèle simple d'estimation de la charge, il est devenu un modèle qui cherche à garantir un progrès du projet à travers un développement en spirale pour consolider l'estimation de la charge, l'architecture du projet et réduire le risque.

D'autre part, la méthode comprend 3 sous modèles :

- Le premier modèle (Application composition) est utilisé pour estimer la charge et le délai de développement dans un contexte de méthodologie RAD et de boîte à outils logiciels.
- Les deux autres modèles sont utilisés dans la même perspective que le premier pour des projets plus classiques ou d'envergure ainsi que les projets intégrés.
 - Le premier des deux derniers modèles (Early Design) permet une première estimation alors que tous les éléments du projet ne sont pas encore connus.
 - le second (Post Architecture) quand l'architecture du projet est connue.

Une redéfinition

L'équation générale du calcul de la charge est la même que dans COCOMO 81 mais les exposants ne sont plus déterminés une fois pour toutes en fonction de la classe des projets mais en fonction d'un exposant corrigé par cinq paramètres : L'expérience, la flexibilité du développement, la maîtrise des risques du projet, la cohésion de l'équipe, la maturité du processus. Quant au facteur d'ajustement, il dépend désormais de 17 paramètres dont 7 nouveaux ce qui signifie que 5 paramètres pour une part jugés dépassés ont été retirés ou fondus dans d'autres.

5.3.2 Grandes évolutions

L'équation générale du calcul de la charge est la même que dans COCOMO 81 mais les exposants ne sont plus déterminés

une fois pour toutes en fonction de la classe des projets mais en fonction d'un exposant corrigé par cinq paramètres :

- l'expérience,
- la flexibilité du développement,
- la maîtrise des risques du projet,
- la cohésion de l'équipe,
- la maturité du processus.

Quant au facteur d'ajustement, il dépend désormais de 17 paramètres dont 7 nouveaux ce qui signifie que 5 paramètres pour une part jugés dépassés ont été retirés ou fondus dans d'autres.

Le facteur *a* dans l'équation de calcul est un facteur linéaire. Pour COCOMO II 1997, il est par exemple de 2,45¹³.

5.3.3 Correspondance

Outre l'estimation directe en fonction du nombre de ligne des programmes¹⁴, COCOMO II permet d'utiliser aussi les points de fonction pour faire l'estimation. Il existe une table de correspondance établie par Caper Jones qui définit la taille du programme en fonction du langage et du nombre de points de fonction.

5.3.4 Exposant du projet

Comme la productivité décroît avec la taille des projets, COCOMO II fixe un exposant lié au type de projet. Cet exposant peut être modifié par l'action des cinq paramètres que nous avons cités ci dessus.

Pour obtenir l'exposant en fonction des paramètres on part de l'équation suivante :

1,01 + somme des paramètres

Les paramètres qui influencent l'exposant sont classés selon une échelle de valeur à 6 positions comme dans COCOMO 81 (Très basse, basse, normale, haute, très haute, extra haute) et notée de 5 à 0¹⁵.

L'expérience mesure le degré de familiarité avec les nouvelles technologies et la spécificité du domaine.

La flexibilité du développement mesure la flexibilité de la conception, c'est-à-dire le potentiel de solutions logicielles alternatives ou qui

¹³ Dans l'esprit de COCOMO ce facteur peut être adapté pour tenir compte du type de projet comme de la culture de l'organisation. D'autre part, ces facteurs linéaires de productivité sont souvent produits par les auteurs et font l'objet de révisions suivant l'adaptation de la méthode. Il existe, en effet, de nombreuses variantes, souvent privées, de la méthode qui souvent multiplient à l'envi les paramètres du facteur d'ajustement.

¹⁴ Par rapport à COCOMO 81, il existe quelques subtilités sur la détermination de la taille (mesure plus précise des instructions, introduction d'un taux d'erreur dans la production du programme, impact de la réutilisation)

¹⁵ Dans le calibrage de la méthode pour un modèle spécifique comme le modèle de post architecture les facteurs sont ajustés. COCOMO II prévoit une évolution régulière des coefficients en fonction des données historiques collectées.

Facteurs d'échelle	Très basse	Basse	Normale	Haut	Très haut	Extra haut
Expérience (PREC)	0,0405	0,0324	0,0243	0,0162	0,0081	0,00
Flexibilité du développement (FLEX)	0,0607	0,0486	0,0364	0,0243	0,0121	0,00
Prise en compte du risque (RESL)	0,0422	3,38	0,0253	0,0169	0,0084	0,00
Cohésion de l'équipe (TEAM)	0,0494	3,95	0,0297	0,0198	0,0099	0,00
Maturité du processus (PMAT)	0,0454	3,64	0,0273	0,0182	0,0091	0,00

permettent de modifier le besoin en cas d'obstacle.

La prise en compte du risque mesure le degré de résolution du risque au sein de l'architecture du projet.

La cohésion de l'équipe mesure la capacité de l'équipe à travailler en commun.

La maturité du processus évalue la maturité de l'organisation de la production logicielle.

5.3.5 Ajustement

Il faut ensuite prendre en compte le facteur d'ajustement qui est gouverné par 17 paramètres.

La valeur de chaque paramètre est multipliée par la suivante.

Si tous les paramètres sont dans la normale, le produit total est 1 et donc les facteurs d'environnement ne modifient pas la charge totale.

Si le produit est supérieur à 1 la charge totale est majorée. Elle est donc minorée si le produit est inférieur à 1

5.3.6 Facteurs d'ajustement liés au produit

RELY : Fiabilité requise

DATA : Taille de la base de données - Se détermine en fonction du rapport de taille de la base de données à la taille des programmes.

CPLX : Complexité du produit. Elle dépend de la complexité des programmes, des interfaces, de la gestion des données, etc.

RUSE : Réutilisation requise - Mesure le degré de réutilisation des programmes requis et donc prend en compte l'effort additionnel pour développer les composants réutilisables.

DOCU : Documentation - Niveau de production de la documentation

5.3.7 Facteurs d'ajustement liés à la plate-forme de développement

TIME : Contraintes liées au temps d'exécution - montant des ressources cpu disponibles

STOR : Contraintes liées à la taille mémoire - taux d'utilisation des mémoires disponibles

PVOL - Volatilité de la plate forme - tient compte des évolutions majeures qui pourraient survenir sur la plate forme.

5.3.8 Facteurs d'ajustement liés au personnel

ACAP - Compétence des analystes - évalue la capacité du personnel à accomplir des tâches analytiques mais aussi à coopérer et communiquer.

PCAP - Compétence des programmeurs.

AEXP - Expérience du domaine d'application - évalue l'expérience de l'équipe sur le type d'application du projet

PEXP - Expérience de la plate forme de développement

LTEX - Expérience des langages et outils de programmation

PCON - Continuité du personnel. Tient compte de la rotation du personnel.

5.3.9 Facteurs d'ajustement liés au projet

TOOL - Utilisation d'outils logiciels

SITE - Développement sur des sites distants - Evalue la proximité physique des membres de l'équipe.

physique des membres de l'équipe.

SCED - Contraintes de délai. Evalue les contraintes de délai imposées à l'équipe projet.

5.3.10 Le tableau des valeurs

Pour chaque facteur des critères sont établis afin de le situer sur l'échelle de valeur qui va de très bas à extra haut. La valeur a priori¹⁶ pour les divers facteurs sont¹⁷ :

Facteurs d'ajustement	Très basse	Basse	Normale	Haut	Très haut	Extra haut
RELY : Fiabilité requise	0,75	0,88	1,00	1,15	1,40	-
DATA : Taille de la base de données	-	0,94	1,00	1,08	1,16	-
CPLX : Complexité du produit	0,75	0,88	1,00	1,15	1,30	1,65
RUSE : Réutilisation requise	-	0,89	1,00	1,16	1,34	1,56
DOCU : Documentation	0,85	0,93	1,00	1,08	1,17	-
TIME : Contraintes liées au temps d'exécution	-	-	1,00	1,11	1,30	1,66
STOR : Contraintes liées à la taille mémoire	-	-	1,00	1,06	1,21	1,56
PVOL - Volatilité de la plate forme	-	0,87	1,00	1,15	1,30	-
ACAP - Compétence des analystes	1,5	1,22	1,00	0,83	0,67	-
PCAP - Compétence des programmeurs	1,37	1,16	1,00	0,87	0,74	-
AEXP - Expérience du domaine d'application	1,23	1,10	1,00	0,88	0,80	-
PEXP - Expérience de la plate forme de développement	1,26	1,12	1,00	0,88	0,80	-
LTEX - Expérience des langages et outils de	1,24	1,11	1,00	0,90	0,82	-

¹⁶ Comme pour les facteurs d'échelle elles sont amenées à varier dans le calibrage des projets.

¹⁷ En anglais on les appelle Effort multiplier values (EM)

programmation						
PCON - Continuité du personnel	1,26	1,11	1,00	0,91	0,83	-
TOOL - Utilisation d'outils logiciels	1,20	1,10	1,00	0,88	0,75	-
SITE - Développement sur des sites distants	1,24	1,10	1,00	0,92	0,85	0,79
SCED - Contraintes de délai	1,23	1,08	1,00	1,04	1,10	-

5.3.11 Une géométrie très variable

Si l'utilisation systématique du coefficient normal ne modifie par l'envergure du projet, le rapport entre l'utilisation systématique du plus petit coefficient et l'utilisation du plus grand donne un rapport de 1 à plus de 3000 !, ce qui laisse uniquement pour ce facteur une belle marge de manœuvre.

5.4. Critique de COCOMO

Tout d'abord on ne résout pas totalement le problème de l'estimation car il reste à estimer la taille selon le nombre de lignes de programme. Cette perspective est évidemment source d'erreur.

Il est également difficile de définir ce qui doit être compté, sans parler des variations d'un langage à l'autre. Comme pour les points de fonction la validation par l'usage de la méthode n'a guère donné de résultats probants.

L'histoire et son impact sur l'évolution de la méthode s'est d'ailleurs chargée de la critiquer (évolution des méthodes de programmation, évolution des langages, utilisation de progiciels et d'outils logiciels, programmation à base de composants réutilisables, etc. qui ont périmé la logique de la programmation traditionnelle et aussi mis l'accent sur l'importance de la conception, de l'organisation, de la qualité, de la gestion de la relation client pour maîtriser un projet).

6. La méthode de Putnam

6.1. Objectif

Cette méthode a pour objectif d'optimiser la charge au cours du développement. Elle n'est donc pas, au sens strict, une méthode de prédiction de la charge et, en ce sens, elle n'entre pas dans le champ de notre étude. Nous la présentons rapidement car elle est souvent associée aux méthodes de mesure. Elle donne, par contre, des réponses sur la répartition de la charge dans le temps et sur la relation entre raccourcissement du projet et augmentation de la charge.

6.2. Exposé de la méthode

6.2.1 *Les travaux de Norden*

Putnam a repris en les adaptant au génie logiciel les travaux que P.V. Norden avait réalisés (en 1958 et 1963) sur les projets de recherche et développement.

Pour Norden :

- l'effectif optimal à un temps t doit être proportionnel à la charge de travail restante ;
- l'efficacité de l'équipe de développement s'améliore avec le temps.

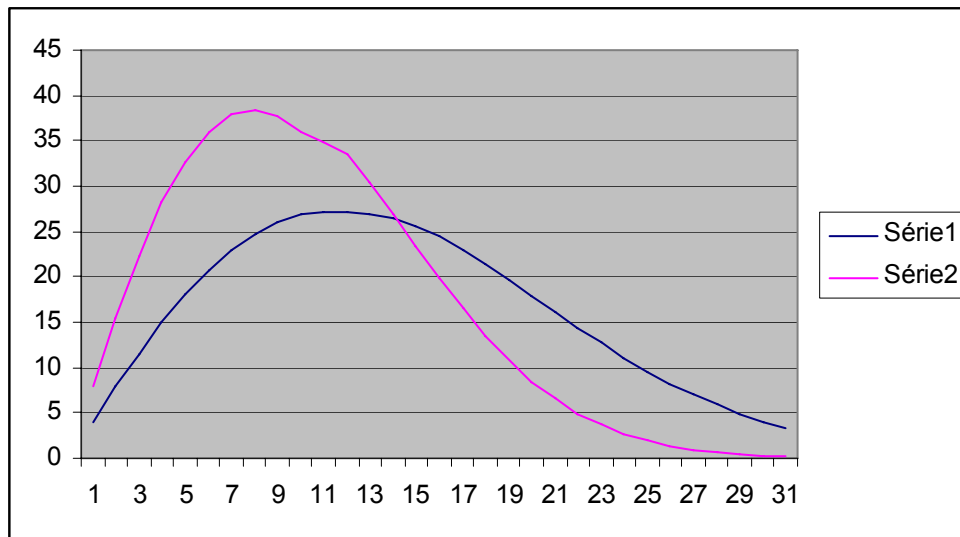
La charge instantanée, la charge au temps t , étant la dérivée de la charge cumulée, on obtient une courbe de distribution de la charge. Cette courbe est une courbe de Rayleigh (du nom du savant anglais, théoricien notamment de l'optique) c'est-à-dire une courbe exponentielle négative.

6.2.2 *Schéma de la charge*

La courbe a donc la forme suivante :

En abscisse : le temps (semaines)

En ordonnée : la charge en ETP



6.2.3 Interprétation de la courbe

L'effort, la charge, est donc réparti en fonction du temps selon une courbe de Rayleigh. Le sommet de la courbe peut être atteint dans des temps variables suivant l'intensité d'un paramètre que Putnam désigne comme représentatif de la complexité du projet. Plus un projet est difficile, plus l'accroissement des effectifs est important.

Selon les constatations de Putnam, le sommet de la courbe des effectifs est atteint au moment de la réception du logiciel. L'effectif maximum comme le temps de développement peuvent être appréciés par rapport au point où la courbe atteint son maximum.

6.2.4 Conséquence

De ces équations on peut donc déduire :

- ▶ l'effectif de pointe une fois estimées la charge totale et la date de réception ;
- ▶ la charge totale du projet quand le maximum des effectifs et le temps de développement sont atteints.
- ▶ l'impact de la variation de la durée de développement sur la difficulté du projet.

6.2.5 Conclusion

A partir des résultats obtenus ci-dessus et d'observations des projets réels, Putnam établit une relation entre la charge totale, le nombre d'instructions et la durée du développement. L'effort à fournir est en relation inverse à la puissance 4 du temps de développement. Ce qui signifie que par exemple un raccourcissement du temps de développement de 10% suppose une augmentation de la charge totale de 40%. Dans le modèle de Putnam, l'augmentation de la durée diminue la charge tandis que son raccourcissement l'accroît. En revanche dans le modèle COCOMO, il y a dans les deux cas un accroissement de l'effort.

7. Conclusion

7.1. Une démarche qui reste à valider

Aucune de ces méthodes n'a recueilli un assentiment général. Elles ont des faiblesses inhérentes à leur conception, et surtout leurs résultats ne sont pas transposables d'une équipe de développement à une autre, d'une industrie à une autre, d'un type de logiciel à un autre, d'un langage de développement à un autre. La validité de l'investissement dans de telles démarches n'a donc pas été probante. On ne peut cependant a priori écarter l'idée qu'il puisse y avoir une méthode simple pour mesurer des systèmes complexes, ce d'autant plus qu'on l'appliquerait à un environnement homogène.

7.2. Un terrain encore à explorer

La métrologie du logiciel demeure cependant un vaste champ de recherche dans lequel l'Europe occupe une part restreinte¹⁸. Seules les méthodes relevant des points de fonction sont engagées dans une démarche (chaotique, comme nous l'avons vu) de normalisation ISO.

En France, le champ de la métrologie n'est pratiquement pas couvert. Il n'existe pratiquement pas de littérature sur ces questions. L'AFNOR s'intéresse plus à la qualité qu'à la mesure. La France participe au sous comité ISO sur le génie logiciel mais n'a pas de représentant dans le comité de travail sur la mesure de la taille fonctionnelle des logiciels.

7.3. Des choix plausibles

Pour les adeptes de la méthode des points par fonction, on peut envisager l'usage de COSMIC-FFP. Cette méthode est cependant sur le plan des concepts beaucoup plus complexe et elle est moins balisée sur le plan opérationnel du fait du faible retour d'expérience. En revanche, elle a l'avantage d'être la plus récente et donc a priori la mieux actualisée. D'autre part, elle est maintenue par une équipe de spécialistes reconnus et une partie de la littérature est accessible en français. Elle n'est pas non plus totalement finalisée, ce qui laisse la possibilité de s'y insérer plus ou moins activement en amont.

Par ailleurs, comme nous l'avons déjà dit, il ne faut pas écarter a priori l'idée d'une adaptation réussie de la méthode dans un contexte donné. Bâtir un référentiel spécifique peut prendre du temps mais c'est une nécessité pour avoir une méthodologie cohérente, sans

¹⁸ La méthode des points de fonction a gagné via Mark II le Royaume-Uni. Les néerlandais ont également adopté une méthode proche. On peut cependant citer la méthode MCP - Méthode de conduite de projet informatiques de M. Gedin et certains projets européens (AMI).

pour autant garantir le succès. De ce point de vue, l'usage de COCOMO II, une fois déterminé les points de fonctions d'un projet, permet de donner rapidement une estimation bâtie sur le référentiel général de la méthode (avec donc toutes les approximations que cela suppose).