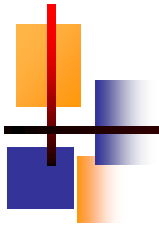
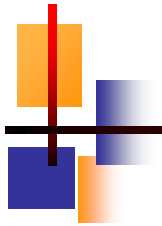


Génie Logiciel (MLI532)

Lamine BOUGUEROUA
Lamine.bougueroua@esigetel.fr



Introduction au génie logiciel
La documentation
Eléments de gestion de projet
Approche UML

Elément( )] TET EMC 8/P <</MCID 12>> W* ETBT/F1 2485f1 187n

Génie logiciel

- Evolution :
  - Les ordinateurs devenaient plus puissants
  - Coût du matériel en diminution
  - La construction des logiciels plus complexe

- Conséquences : → La crise du logiciel

  La fabrication des logiciel trop chère

  Dépassement du budget

  Ergonomie discutable

  Les performances faibles

  Applications non évolutives

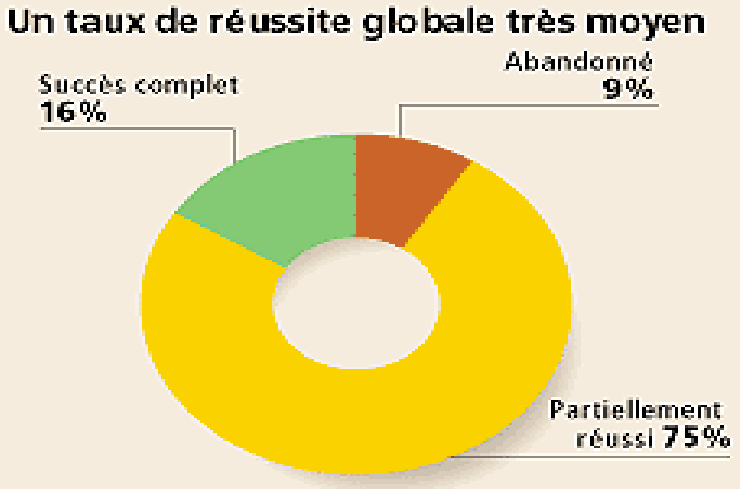
- le génie logiciel

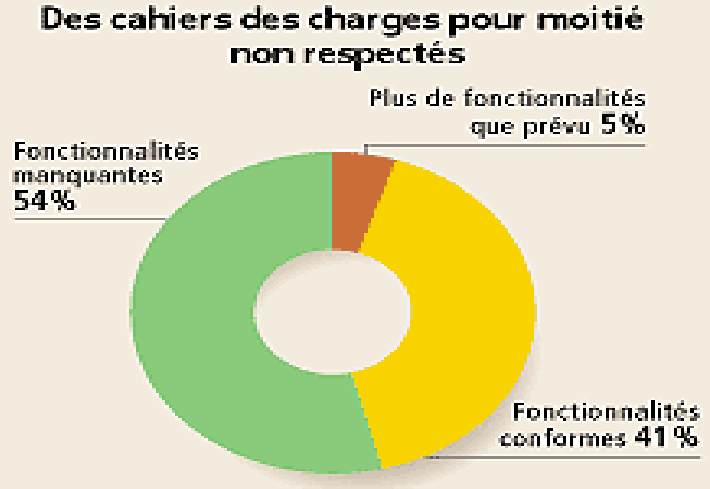
Génie logiciel

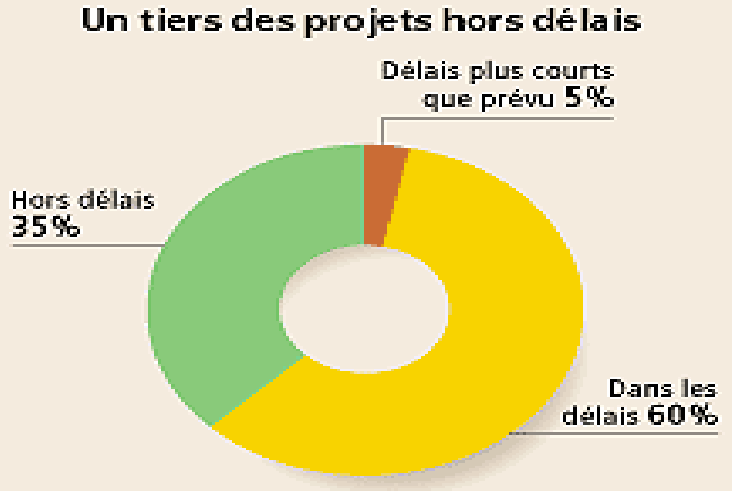
Ex. projets informatique menés en Angleterre

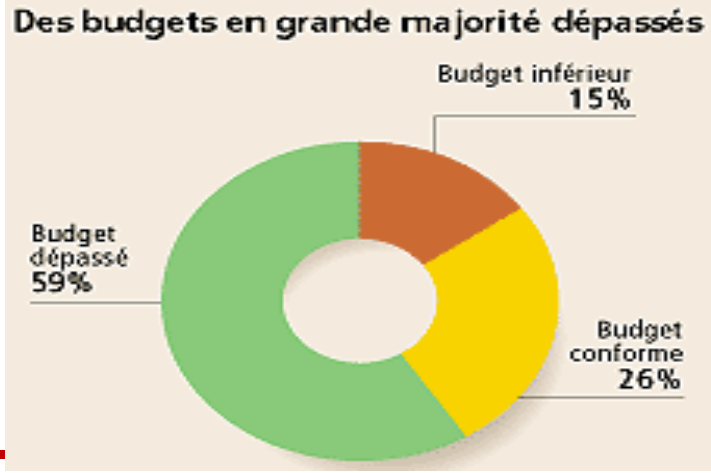
Rapport du Royal Academy of Engineering et la British Computer Society (BCS) - 2003.

http://www.01net.com/article/243555.html









Génie logiciel

Définition : l'ensemble des activités de conception et

tendant à rationaliser la production du logiciel et son suivi . ***(journal officiel 1984)***

Autrement dit :

- Le génie logiciel désigne :
  - Les outils
  - Les méthodes
  - Les techniques
  - Les normes
  -
-

Génie logiciel la norme

- Projet SWEBOK (*Software Engineering Body of Knowledge*):

- Lancé par IEEE Computer society

- Regroupe plusieurs industriels et universitaires

- Définit plusieurs domaines de connaissances pour le génie logiciel

- Définit une liste des disciplines reliées au génie logiciel

Génie logiciel la norme

- Les disciplines reliées au génie logiciel

✓

✓ Science Informatique

✓ Mathématiques

✓ La gestion de projet

✓

✓

✓

Génie logiciel la norme

- Domaines de connaissances pour le génie logiciel

- ✓ Les exigences du logiciel
- ✓ La conception du logiciel
- ✓ La construction du logiciel
- ✓ Les tests logiciels
- ✓ La maintenance du logiciel
- ✓ La gestion de configuration du logiciel
- ✓ *software engineering management*)
- ✓ *software engineering process*)
- ✓ *software engineering tools and methods*)
- ✓ L'assurance qualité du logiciel

Génie logiciel

Définition :

Les étapes :

- Planification du projet
- Analyse de besoin
- La spécification : fonctionnelle, technique
  - Conception
  - Le développement
  - Tests : unitaire, intégration, validation

- Recette : validation
- Maintenance et évolution : exploitation

➔

- Analyse de besoin :

Objectif : définir une première version du cahier de charge

- Etudier la faisabilité
- Etablir le plan général
- Faire une estimation approchée du coût et des délais de réalisation
- Effectuer une enquête au niveau des demandeurs de

les concernant

- Analyse de besoin (Etude de Faisabilité):

➢ Déterminer si le développement proposé vaut la peine d être mis en œuvre, compte tenu de attentes et de la difficulté de développement

Etude de marché : Déterminer s il existe un marché potentiel pour le produit.

- Spécification :

Objectif : définir un cahier de charge (fonctionnalités, contraintes, ressources et planning)

- Décrire le modèle fonctionnel
- Faire une estimation précise du coût et des délais de réalisation ainsi que les ressources utilisées
- Donner un planning de développement

- Spécification :

Déterminer comment on va développer le logiciel

- Analyse des coûts : établir une estimation du prix du projet
- Planification : établir un calendrier de développement
- Assurance qualité du logiciel : déterminer les actions qui permettront de s assurer de la qualité du produit fini
- Répartition des tâches : hiérarchiser les tâches et sous-tâches nécessaires au développement du logiciel

- Conception :

Objectif : modéliser le logiciel (deux phases : générale et détaillée)

- Décomposition en plusieurs modules
- Description de chaque module : rôle, lien avec les autres
- Description détaillée de chaque module
- Préparation des solutions en code intermédiaire (ex. pseudo code)

- Conception :

1. Conception générale
   - Conception architecturale : déterminer la structure du système
   - Conception des interfaces : déterminer la façon dont les différentes parties du système agissent entre elles

2. Conception détaillée : déterminer les algorithmes pour les différentes parties du système

Génie logiciel

- Développement (Codage):

Objectif : implanter les solutions sur machine

- Traduction des algorithmes en langage machine
- Préparation de la documentation technique

- Editeur de texte
- Compilateur
- Débogueur
-

Génie logiciel

- Tests :

Objectif :

avec le cahier de charge

- Préparer des jeux de tests correspondant aux fonctionnalités du logiciel.
- Faire des tests :
  - sur chaque module séparément
  - après assemblage des modules
  - globale en vérifiant toute les fonctionnalités
- Rédaction de la documentation pour les utilisateurs

Génie logiciel

- Tests :

Essayer le logiciel sur des données d exemple pour s assurer qu il fonctionne correctement

- ✓ Tests unitaires : faire tester les parties du logiciel par leurs développeurs
- ✓ Tests d intégration : tester pendant l intégration
- ✓ Tests de validation : pour acceptation par l acheteur
- ✓ Tests système : tester dans un environnement proche de l environnement de production
- ✓ Tests Alpha : faire tester par le client sur le site de développement
- ✓ Tests Bêta : faire tester par le client sur le site de production
- ✓ Tests de régression : enregistrer les résultats des tests et les comparer à ceux des anciennes versions pour vérifier si la nouvelle n en a pas dégradé d autres

- Recette :

Objectif :
ou de livraison

- Préparation de la version livrable
- Le client devient propriétaire du produit
- Dernière étape du projet (signature du procès-verbal et déclenchement de la période du garantie)

- Mise en disposition du produit final
- Fin du contrat

Génie logiciel

- Recette :

- Fournir au client une solution logicielle qui fonctionne correctement

Installation : rendre le logiciel opérationnel sur le site du client

Formation : enseigner aux utilisateurs à se servir du logiciel

Assistance : répondre aux questions des utilisateurs

- Maintenance :

Objectif : Adopter et améliorer le logiciel

- Considérer comme étant la plus longue phase
- Plusieurs types de maintenance :
  - Corrective : correction des erreurs non détectées par les tests
  - Adaptative : adaptation du produit aux nouvelles contraintes
  - Evolutive : amélioration et optimisation des performances

- Maintenance :

✓ Mettre à jour et améliorer le logiciel pour assurer sa pérennité

✓ Pour limiter le temps et les coûts de maintenance, il faut porter ses efforts sur les étapes antérieures

	Répartition effort dév.	Origine des erreurs	Coût de la maintenance
Définition des besoins	6%	56%	82%
Conception	5%	27%	13%
Codage	7%	7%	1%
Intégration Tests	15%	10%	4%
Maintenance	67%		

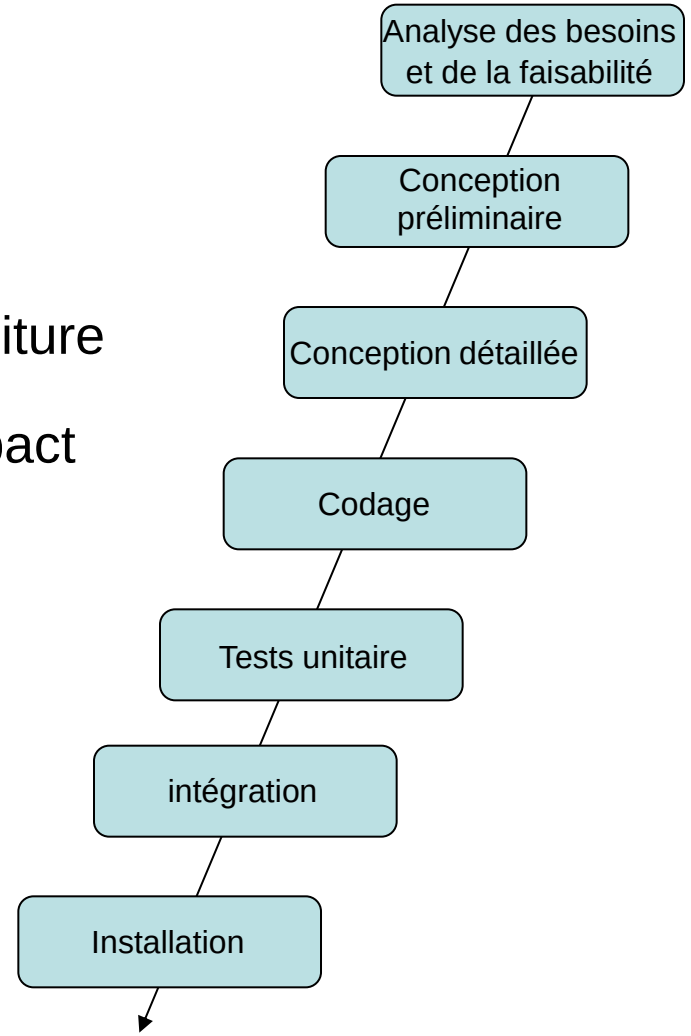
- Le développement d'un logiciel se fait suivant un cycle appelé le cycle de vie du logiciel

- Le cycle de vie est décomposé en phases de développement.

- logiciel :
  - Modèles linéaires
  - Le cycle en cascade
  - Le cycle en V
  - Modèles non linéaires
  - Le cycle itératif
  - Le cycle en spirale

- **Cycle en cascade** :

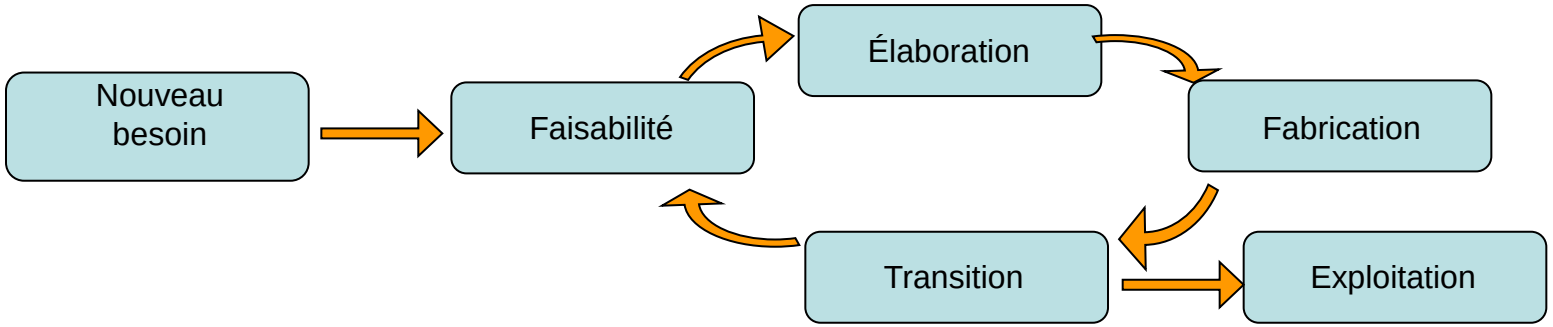
- Hériter de celui du bâtiment.
- Il date des années 70
- Construire les fondations avant la toiture
- Une modification en amont a un impact important sur les coûts en aval
- Développement étape par étape



Adapté pour des projets de petite taille, et dont le domaine est bien maitrisé

Cycle itératif :

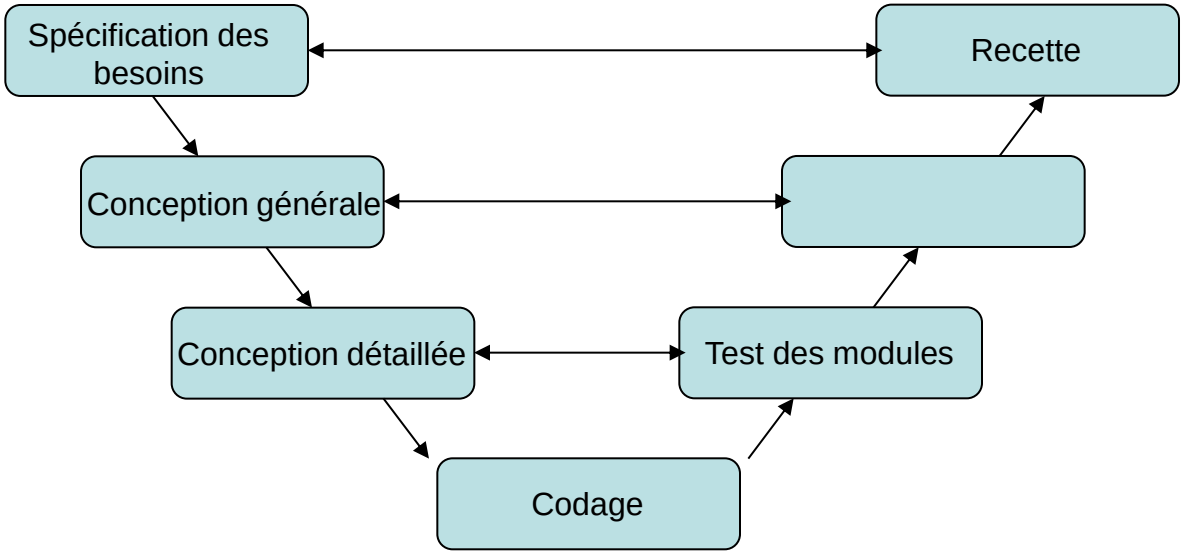
- Accepter un nouveau besoin (faisabilité)
- Faire la conception (élaboration)
- Passer à la réalisation (fabrication)
- Préparer la livraison au client (transition)



Meilleure intégration du client dans la boucle, produit conforme à ses attentes

Le cycle en V :

- Deux parties : montante et descendante
- Limiter les retours aux étapes précédentes
- Standard depuis les années 80



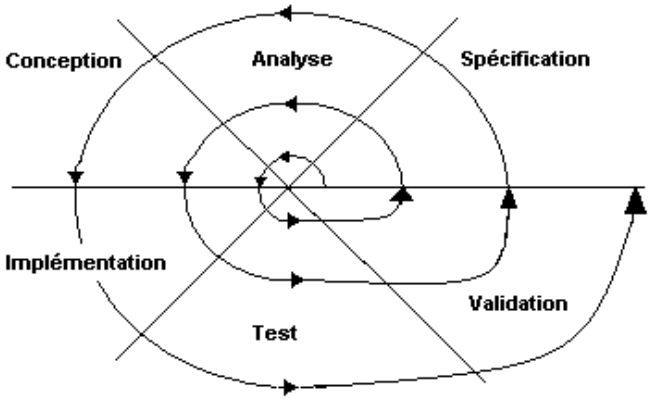
Adapté pour des projets dont le domaine est bien maitrisé

- Ces phases sont échelonnées dans le temps
- Chaque phase se termine par la remise d'un (ou plusieurs) document(s) validé(s) conjointement par l'utilisateur et le développeur
- Une phase de développement se termine lorsque la revue de cette phase est faite
- Une phase ne peut commencer que lorsque la précédente est terminée
- A la fin de chaque phase, l'utilisateur et le développeur sont d'accord

Remarque : La décomposition en phases de développement permet donc le suivi du projet par l'utilisateur

Le cycle en Spirale :

- Définit par Barry Boehm en 1988
- Les étapes sont identiques à ceux du cycle en V
- Implémentation de versions successives
- Les risques sont évalués à chaque cycle
- Un cycle est décomposé en étapes



Meilleure maitrise des risques, mais nécessite une très grande expérience

- Utiliser dans le cas des projets grands, chères et complexes

Exemple : *US military - Future Combat Systems*

- **Démarche :** succession

  Mieux maitriser le déroulement

  Meilleure visibilité pour les utilisateurs sur certains résultats

  Intermédiaires et garantir que le résultat final sera celui attendu

Formalisme : défini par:

Un langage formel

Un langage semi-formel généralement graphique

Un langage naturel

Fonction :

Représenter le monde réel perçu par le concepteur

Outil de communication entre informaticiens et utilisateurs

Constitué par un ensemble de modèles

Une bonne compréhension des besoins des utilisateurs

- Modèles :

- Représentation abstraite de la réalité qui exclut certains détails du monde réel

- Permet de réduire la complexité phénomène en éliminant les détails significatif

- Reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé, les limites du phénomène modélisé dépendent des objectifs du modèle

Documents produits durant le cycle de vie

Cahier des charges	
Calendrier du projet	Planification
Estimation des coûts	Planification
Plan d'assurance qualité	Planification
	Spécifications
Plan de test	Spécifications
Manuel utilisateur préliminaire	Spécifications
Conception architecturale	Conception
Conception détaillée	Conception
Code source et documentation	Implémentation
Manuel utilisation final	Implémentation
Rapport des tests	Tests

Les documents de référence dans un modèle du cycle en V:

Besoins et Faisabilité	Spécification	Conception générale	Conception détaillée	Codage	Test unitaire	Test d'intégration	Test de validation	Recette
Spécification de besoins utilisateur								Procès verbal de validation
	-Cahier des charges -Spécifications générales - Spécification technique des besoins						Rapport	
		- Document de définition du logiciel - Plan de test				Rapport		
			Rapport		Rapport			
				Code source				

- Les différents acteurs :

-

-

-

-

- Equipe de développement : conception détaillée + codage + tests unitaires

La conception de programmes (1/3)

deux grandes étapes qu'il est nécessaire de dissocier:

❑ La première, très en amont, ressemble à une recherche ou à une étude de faisabilité du projet:

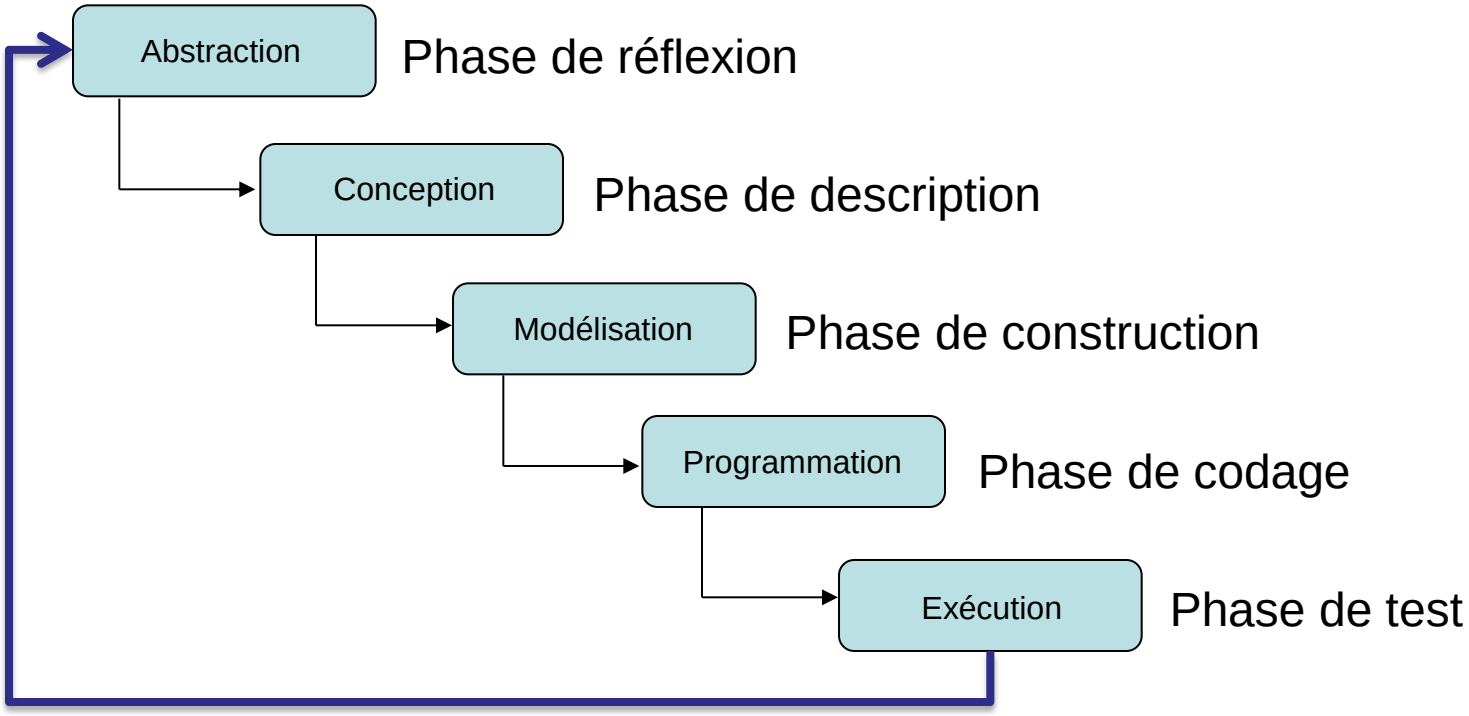
- ✓ L'utilisateur a seulement une idée floue de son besoin et surtout ne sait pas comment y répondre
- ✓ Il ne sait pas quelle solution est faisable
- ✓ pouvoir les valider ou les invalider
- ✓ Les systèmes experts y sont en général utilisés car adaptés au besoin
- ✓ Dans les entreprises, ce sont les départements d'études ou de recherche qui effectuent ce genre de travail

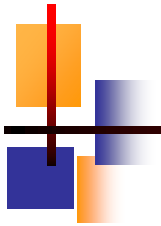
La conception de programmes (2/3)

- La seconde, plus en aval, est un développement plus concret qui ne pose pas de problème de faisabilité:

  - ✓ Il est possible de définir clairement le besoin auquel le logiciel répond
  - ✓ On sait d'avance qu'il existe une solution faisable pour réaliser le logiciel
  - ✓ Le savoir-faire dans ce domaine peut être formalisé
  - ✓ Dans les entreprises, ce sont les départements de développement de logiciels opérationnels qui effectuent ce genre de travail

La conception de programmes (3/3)



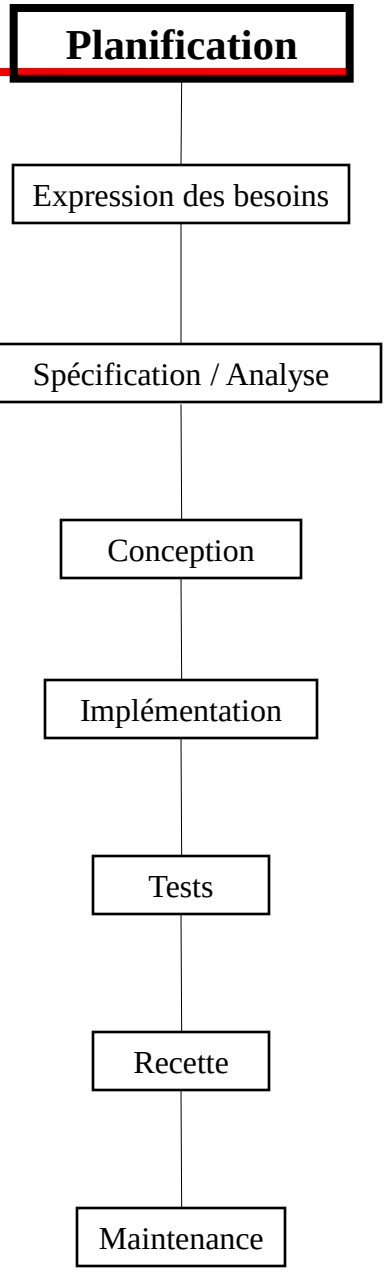


Estimation par abaque

Estimation imposée

COCOMO

Les points de fonctions



Estimation par abaque

- La méthode la plus employée, et la plus fiable
- Consiste à identifier les tâches dont on souhaite évaluer la charge dans des tableaux issues de projets analogues
- Peut introduire des facteurs correctifs liés à la :
  - Complexité de la tache
  -
  - Qualification du personnel sollicité

	Débutant	Débutant	Confirmé	Confirmé
	Simple	Complexe	Simple	Complexe
Tâche 1	2 j	4	1	2
Tâche 2	1j	2	1	1,5

Estimation par abaque

- déclinent en introduisant de nombreux facteurs :
  - Architectures technique standard
  - Services Standard
  -
  - Options fonctionnelles
  - Options techniques
  -
- projets

Estimation imposée

- eux-mêmes imposés
  - cost);
  - market);
  - Et on adopte un rétro-planning
- rôles :
  - Devancer un concurrent
  - Dynamiser les équipes
  - Respecter des règles de gestion strictes
- imposée e

Estimation imposée

- Ajouter le facteur risque
  - Les tâches dont la réalisation comporte un risqué élevé peuvent être réduites : sous-traitance, composant prêt à

Tâche	Charge (Jour/H)	Risque	Décision
Composant de reconnaissance de caractères	400	Performances insuffisantes	Pour la version 1.0, remplacer par composant OCR
Support de polices « petites »	50	médiocre	Supprimer

- Définir une productivité > 100% !!!
  - organisationnels pour économiser des jours
  - Ajuster la méthode plus que le fond

COnstructive COst Model (COCOMO)

- Naissance :
  - 1981 (Barry Boehm), COCOMO 81
- Intérêt :
  - estimer l'effort à fournir dans un développement logiciel
  - estimer la durée du projet et le nombre de personnes requises
- L'estimation :
  - peut intervenir à différentes étapes du projet
  - se base sur le nombre de lignes de code
- Le Modèle
  - basé sur une étude statistique (estimation analytique)
  - 63 projets (2k 100k lignes

COnstructive COst Model (COCOMO)

Quelques règles d'estimation

- Prendre le temps pour la réflexion (éviter de deviner)
- Prévoir du temps pour l'estimation et la planifier
- Utiliser des données de projets précédents
- Prendre l'avis des acteurs participants au projet (ex. développeurs)
- Estimer par consensus (consulter les différentes équipes)
- Proposer des dates au plus tôt et au plus tard
- Estimer par difficultés (facile, moyen, difficile)
- Ne pas oublier les tâches récurrentes : ex. documentation, démo., formation, intégration à l'existant, récupération des données, réunions, gestion qualité, absentéisme (congés, maladie, RTT
- Utiliser plusieurs techniques d'estimation
- Prendre en compte les risques de gestion dans l'estimation

COCOMO II :

Il existe trois grandes phases dans le processus d'estimation

- Estimer la taille du produit (ex. nombre de lignes de codes)
-
- Estimer la durée en mois / semaines

-

$$\text{Effort nominal} = A * KLOC^{B}$$

- Unité : nombre de homme mois (HM) [1 HM correspond à 152heures]
- KLOC (Kilo Lines Of Code) : nombre de milliers d'instructions.
- A = 2,94 (productivité nominative)
- 1.26), Valeur nominale pour B = 1,16

COCOMO II

Barry Boehm en 1995 propose d'encadrer les prédictions par une fourchette optimiste/pessimiste.

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

Pour utiliser les facteurs multiplicatifs du tableau, il suffit de multiplier le point d'estimation unique par les coefficients

COCOMO II :

- Modèle de Base : $HM\ Base = A * KLOC^{B}$

  - Exemple :
  -
  - A = 2,94 (productivité nominative)
  - B = 1,0997

  - HM Base = 28,9 Homme / Mois

COCOMO II : détermination du facteur B

- Les facteurs à prendre en compte :
  - PREC (connaissances préalable) : mesure la compréhension des objectifs du produit, s'il y a une expérience sur des logiciels proches, s'il y a un besoin d'architecture innovante.
  - FLEX (Flexibilité (liberté) vis-à-vis de spécifications) : mesure s'il faut se conformer avec des spécifications ou pas.
  - RESL (bonne gestion des risques) : mesure s'il y a un plan de gestion des risques, le % du temps du projet consacré à établir
  - TEAM (bonne cohésion de l'équipe) : mesure la cohésion entre les intervenants sur le projet : utilisateurs, client, développeurs,
  - PMAT (bonne maturité des processus de développement) : mesure

COCOMO II : le calcul du facteur B

Valeur nominale 1,16	1,15						
	-	**Très**					

COCOMO II :

- Modèle de Base : $HM\ Base = A * KLOC^{B}$

- Modèle avec ajustement : $HM = HM\ Base * Aj$

Aj : 17 facteurs de productivité)

- Temps de développement : $Tdev = C * HM^{D}$

C : constante du modèle

D : échelle appliquée au temps de développement

COCOMO II : ajustement de l'effort nominal

- L'effort nominal peut-être ajusté à l'aide de 17 facteurs :
  - *3 facteurs liés au projet*
  - *5 facteurs liés au logiciel à développer*
  - *3 facteurs liés à la plateforme d'exécution*
  - *6 facteurs liés au personnel*

- L'ajustement peut-être calculé :
  - au début du projet
  - après que l'architecture ait été fixée

Où Effort nominal = $A * KLOC^{B}$

COCOMO II :

Facteurs d'ajustement après architecture (1/8)

- 1er type de facteurs à prendre en compte :

  les facteurs liés au projet

Facteurs liés au projet	
TOOL	Mesure l'apport de l'utilisation d'outils
SITE	Mesure les impacts du travail sur plusieurs sites de l'équipe de projet
SCED un projet développé selon un calendrier accéléré nécessite plus d'efforts qu'un projet développé sur son calendrier optimal	Mesure les contraintes de planning imposées à l'équipe de projet qui développe le logiciel

Facteurs d'ajustement après architecture (2/8)

- Valeurs d'ajustement pour *les facteurs liés au projet*

(choisir une valeur par facteur) :

TOOL	Edition, codage, débogage	simple	pour les bases du cycle de vie	forte	forte et très intégrée dans le cycle de vie		
	1,24	1,1	1	0,91	0,82		
SITE	International, peu téléphone, courrier papier	multiple villes, multiple entreprises, téléphone, fax	multiple villes, multiple entreprises, email	même ville ou bonne desserte, Com. électronique	même complexe	en un seul endroit, multimédia interactif	en un seul endroit, multimédia interactif
	1,24	1,1	1	0,91	0,82	0,6	0,6
SCED	75% nominal	85% nominal	100% nominal	130% nominal	160% nominal		
	0,82	0,91	1	1,04	1,1		

Facteurs d'ajustement après architecture (3/8)

- 2ème type de facteurs à prendre en compte :

  les facteurs liés à la plateforme

Facteur	
TIME	Mesure la contrainte imposée sur le temps d'exécution du logiciel à développer
STOR	Mesure le pourcentage du stockage disponible utilisé par le logiciel
PVOL	Mesure la complexité du logiciel et du matériel qui est utilisé par le logiciel en cours de développement (le développement d'une base de données par exemple s'appuie sur du matériel et un système d'exploitation)

Facteurs d'ajustement après architecture (4/8)

Valeurs d'ajustement pour *les facteurs liés à la plateforme*

(choisir une valeur par facteur) :

TIME			50% du temps de calcul	70,00%	85,00%	95,00%	
			1	1,11	1,3	1,66	
STORE			50% d'utilisation du stockage	70,00%	85,00%	95,00%	
			1	1,06	1,21	1,56	
PVOL Volatilité de la plateforme		changements majeurs tous les 12 mois, mineurs tous les 1 mois	Majeur : 6 mois, mineur : 2 semaine	Majeur : 2 mois, mineur : 1 semaine	Majeur : 2 semaine, mineur : 2 jours		
		0.87	1	1,15	1,3		

Facteurs	
RELY	Défini le degré de fiabilité attendue de l'application à développer
DATA	Mesure l'effet d'une quantité importante de données à gérer (car cela entraine un temps important consacré aux tests)
CPLX	Mesure la complexité de l'application à développer en terme de : contrôles, calculs, utilisations de périphériques, de gestion des données, d'interface utilisateur
RUSE	Mesure l'effort additionnel à produire pour que l'application développée puisse être réutilisée

Facteurs d'ajustement après architecture (6/8)

- Valeurs d'ajustement pour les *facteurs liés au produit* :

RELY	faible	Pertes sans conséquence	Pertes avec conséquence	perte financière importante	risque pour la vie humaine	
	0,75	0.88	1	1,15	1,4	
DATA		[taille base / taille prog.] < 10	10 < [taille base / taille prog.] < 100	100 < [taille base / taille prog.] < 1000	[taille base / taille prog.] >= 1000	
		0,94	1	1,08	1,16	
CPLX	très faible	faible	nominale	élevée	très élevée	très très élevée
	0,7	0,85	1	1,15	1,3	1,65
RUSE		non	entre projet	entre programme	entre des lignes de produits	entre + lignes de produits
		0,87	1	1,15	1,3	1,4
DOCU	+ étapes projet non couvertes	quelques étapes projet non couvertes	bonne couverture	couverture excessive	couverture très excessive	
	1,23	1,11	1	0,91	0,81	

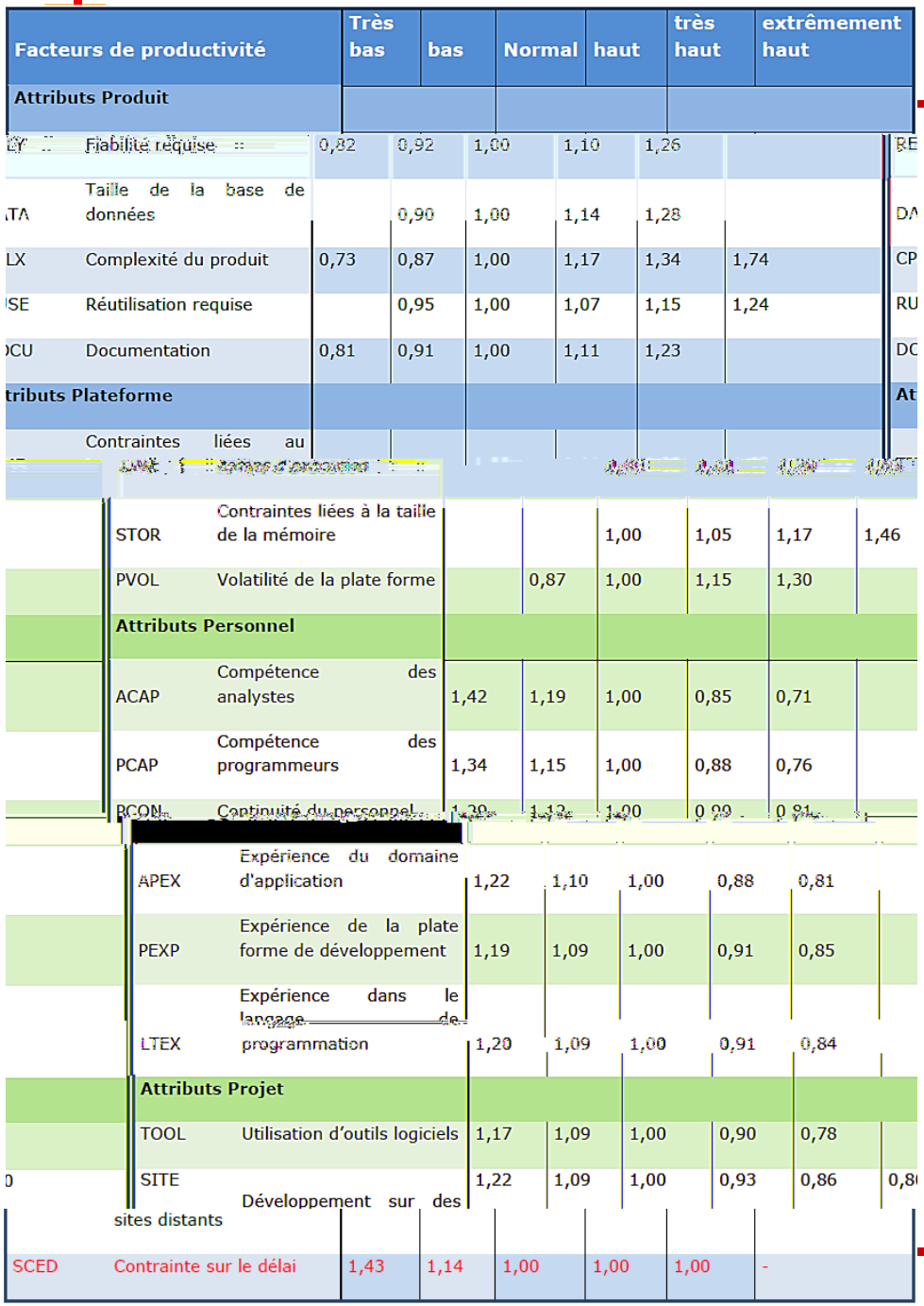
Facteurs d'ajustement après architecture (7/8)

- 4ème type de facteurs à prendre en compte : *les facteurs liés au personnel*

Facteurs	
PCAP	Ce facteur évalue les capacités du développeur en relation avec les autres membres de l'équipe de projet en terme de minutie, d'efficacité... mais aussi capacité à communiquer et collaborer
ACAP	Idem que le facteur PCAP mais appliqué à l'analyste qui travaille sur le projet
PCON	Mesure le turnover annuel dans l'équipe de projet
AEXP	Mesure le degré d'expérience des applications de l'équipe de projet
PEXP	Mesure la compréhension dans l'utilisation de plateforme de plus en plus complexe (interface utilisateur, bases de données, accès réseau...)
LTEX	Mesure l'expérience de l'équipe : langages de programmation et outils logiciel

Facteurs d'ajustement après architecture (8/8)

ACAP	très faible	faible	nominale	élevée	très élevée	Extrêmement élevée
	1,46	1,19	1	0,86	0,71	0,6
PCAP	très faible	faible	nominale	élevée	très élevée	très très élevée
	1,42	1,17	1	0,86	0,7	0,6
PCON	48% par an	24% par an	12% par an	6% par an	3% par an	
	1,42	1,17	1	0,86	0,7	
AEXP	<= 2 mois	6 mois	1 an	3 ans	6 ans	
	1,29	1,13	1	0,91	0,82	
PEXP	<= 2 mois	6 mois	1 an	3 ans		
	1,21	1,1	1	0,9		
LTEX	<= 2 mois	6 mois	1 an	3 ans		
	1,14	1,07	1	0,95		



Facteurs de productivité		Très bas	bas	Normal	haut	très haut	extrêmement haut
Attributs Produit							
	Fiabilité requise	0,82	0,92	1,00	1,10	1,26	
	Taille de la base de données		0,90	1,00	1,14	1,28	
LX	Complexité du produit	0,73	0,87	1,00	1,17	1,34	1,74
SE	Réutilisation requise		0,95	1,00	1,07	1,15	1,24
CU	Documentation	0,81	0,91	1,00	1,11	1,23	
tributs Plateforme							
	Contraintes liées au						
	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	0,8
SCED	Contrainte sur le délai	1,43	1,14	1,00	1,00	1,00	-	

Le tableau des facteurs

COCOMO II : estimation du temps de développement

- Une estimation du temps de développement à partir de l'effort nominal peut-être calculée :

$$\text{Temps de développement} = [3{,}67 * \text{Effort1}^{(0{,}28 + 0{,}2 \times (B-1.01))}]$$

  - où Effort1 est l'effort en mois homme sans prendre en compte le facteur SECD.
  - B est le facteur d'échelle.
  - L'unité du temps de développement est en mois.

Les points de fonction

- Le concept :
  - unité pour mesurer la taille fonctionnelle d'un système d'information vu par les utilisateurs
  - mesure indépendante des techniques de réalisation et d'exploitation du système
  - largement utilisée dans les grands projets informatique

- Buts :
  - estimer les charges de développement
  - mesurer la taille des applications d'une entreprise
  - réaliser des bilans de projets.

Les points de fonction

- 5 étapes
  - Définir le contexte : périmètre, phase de chiffrage, type de projet, les acteurs
  - Identifier les composants: les données manipulées (internes, sortie, interrogation)
  - Calculer pour chaque composant le nombre de points de fonction associé
  - Evaluer
  - Calculer le nombre de PF net

Les points de fonction

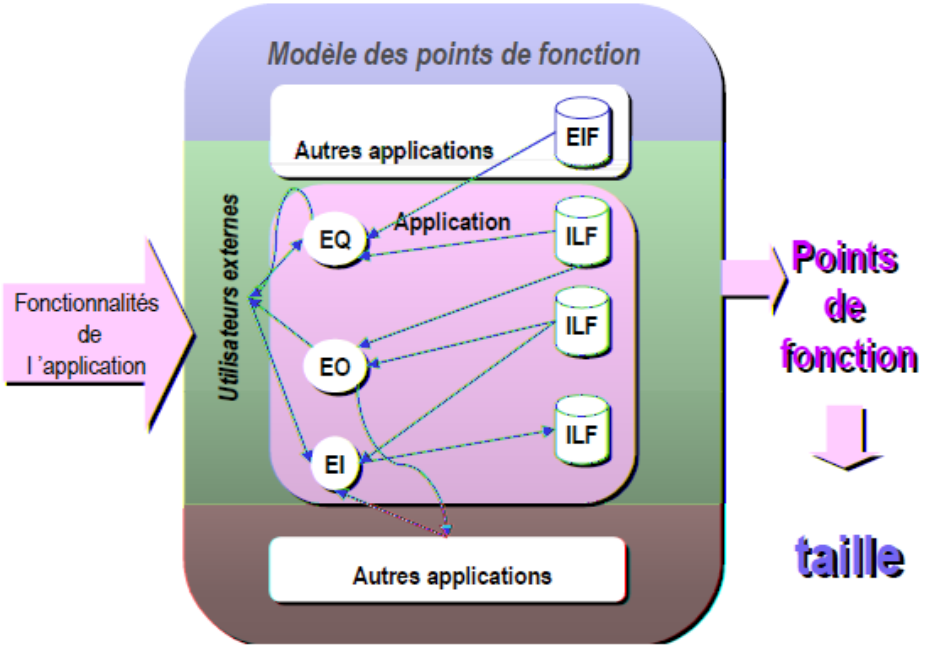
- Définition des points de fonctions :

Fichier Internes Logique	ILF	Groupement de données utilisateur ou d'informations de contrôle utilisés ou générés à l'intérieur du logiciel
Fichier d'Interface Externe	EIF	Fichier passé ou partagé entre programmes
Entrée	EI	Chaque données unique d'utilisateur ou de contrôle qui entre dans le logiciel et ajoute ou modifie des données dans un fichier logique interne
Sortie	EO	Chaque donnée unique d'utilisateur ou de contrôle qui quitte le logiciel
Intérogation Externe	EQ	Chaque combinaison d'entrée/sortie où les entrées génèrent une sortie immédiate (une question)

Les points de fonction

- La taille fonctionnelle du système

= somme des points (ENT, SOR, INT, …)



- Quelques exemples de tailles fonctionnelles
  - portail de vente de pièces détachées sur internet : env. 6 000 points ;
  - logiciel de navigation, type Carminat, Garmin, ... : 1 000 points ;
  - logiciel de comptabilité, type Ciel Compta : 2 000 points ;
  - logiciel de gestion des nomenclatures industrielles : 4 000 points ;
  - logiciel d'analyse de temps et de la paie dans une grande entreprise : 5 000 points.

* Wikipédia http://fr.wikipedia.org/wiki/Point_de_Fonction

Les points de fonction : comment les calculer ?

1°/ Compter les points de fonction pour chaque type :

Fichier Internes Logique	...
Fichier d'Interface Externe	...
Entrée	...
Sortie	6
Interrogation Externe	...

2°/ Pour chaque type de points, les répartir dans un tableau qui mesure sa complexité :

Sorties			
	champs	de	Données
	1 - 4	5 - 15	> 16
0 ou 1	faible	faible	moyen
	1		
2 - 3	faible	moyen	élevé
		2	
> 3	moyen	élevé	élevé
	3		

3°/ Pondérer les points de fonctions du tableau 2 par les poids suivants :

	faible	moyen	élevé
Fichiers Internes Logique	7	10	15
Fichiers d'interface externe	5	7	10
Entrées	3	4	6
Sorties	4	5	7
Interrogations Externe	3	4	6

4°/ Calculer le nombre de points :

Les points de fonction

- Il y a plusieurs tableaux de complexité (voir point 2°/ du transparents précédent) classés par type de points de fonction :

Nombre d'entités logiques (fichiers, écrans...)

Nombre de données

Sorties Interrogation externes			
	1 - 5	6 - 19	> 20
0 ou 1	faible	faible	moyen
2 - 3	faible	moyen	élevé
> 4	moyen	élevé	élevé

Entrées			
	1 - 4	5 - 15	> 16
0 ou 1	faible	faible	moyen
2 - 3	faible	moyen	élevé
> 3	moyen	élevé	élevé

Fichiers internes logiques Fichiers d'interface externe			
	1 - 19	2 - 50	> 51
1	faible	faible	moyen
2 - 5	faible	moyen	élevé
> 6	moyen	élevé	élevé

Les points de fonction

- Une estimation du nombre de lignes de code peut-être obtenue à partir du nombre de points de fonction d'après le langage utilisé :
- unité : ligne de code / point de fonction

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

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
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	91	[illegible]	6
[illegible]	49	[illegible]	80
[illegible]	4	[illegible]	107
[illegible]	320	[illegible]	1
[illegible]	64	[illegible]	1
[illegible]	107	[illegible]	1
[illegible]	71	[illegible]	1
[illegible]	20	[illegible]	1
[illegible]	64	[illegible]	29
[illegible]	15	[illegible]	34
[illegible]	53		

L'ajustement des points de fonction

- Un facteur d'ajustement peut être utilisé (attention ! l'utilisation de ce facteur fait débat) :

Valeurs entre 0 et 5

- Facteur d'ajustement =

- Nombre de points final = facteur d'ajustement x nombre de points non ajusté

	Données de communication
	Fontions distribuées
	Performance
	Configuration lourde
	Taux de transactions
	Saisie des données en ligne
	Conçu pour l'efficacité de l'utilisateur
	Mise à jour en ligne
	Traitement complexe
	Utilisable dans d'autres applications
	Facilité d'installation
	Facilité d'utilisation
	Usage multi-sites
	Facilite le changement