CENTRE DE
MAITRISE DES
SYSTEMES ET
DU LOGICIEL

Les principes du modèle d'estimation COCOMO

Les trois niveaux du modèle

Les limites du modèle

Les origines du modèle

- Dans les années 70s **constat** par de nombreux experts d'une **forte corrélation** entre le **volume** de programmation (exprimées en KLS), l'**effort** de développement et la **durée** du développement
  - L'industrie naissante du logiciel est, à cette époque, une « usine » à développer → pas de progiciel
    - ✓ i.e. la « software factory »
    - ✓ Un acteur clé → le programmeur (et son environnement de programmation)
  - La production de logiciels concerne les constructeurs d'ordinateurs et les grands systèmes de défense
    - ✓ OS IBM 360, BULL GCOS7 et 8, VAX VMS de DEC, etc.

Historique du modèle

- Première version : fin 70s – début 80s ; Publication du livre de B.Boehm *Software engineering economics*, en 1981
  - Très ciblé sur le processus de développement – Contient tous les justificatifs du modèle
- *Ada COCOMO and the Ada process model*, B.Boehm, W.Royce, en 1989
  - ✓ Voir également W.Royce, *Software project management*, 1998.
- *Software cost estimation with COCOMO II*, B.Boehm & al., en 2001
  - Extensions et révisions des facteurs d'influence – Couverture complète du cycle système

Élaboration du modèle COCOMO

- Résulte d'une **analyse**, par des techniques statistiques, des données d'un très **grand nombre de projets** conduite par B.Boehm
  - ✓ C'est fondamentalement un **modèle statistique**

- Classification
  - ❑ **Typologie** des programmes selon la difficulté de réalisation (complexité) en 3 catégories S / P / E
  - ❑ **Phases** de développement
    - ✓ Découpage **temporel** par activité dominante
  - ❑ **Processus** de développement
    - ✓ Découpage en **activités** selon les principes admis à l'époque (modèle en cascade, cycle en V, etc. reconnus dès la fin des 60s)
- Facteurs d'influence sur la **productivité** et le **rendement** des équipes de développement

Principes fondamentaux

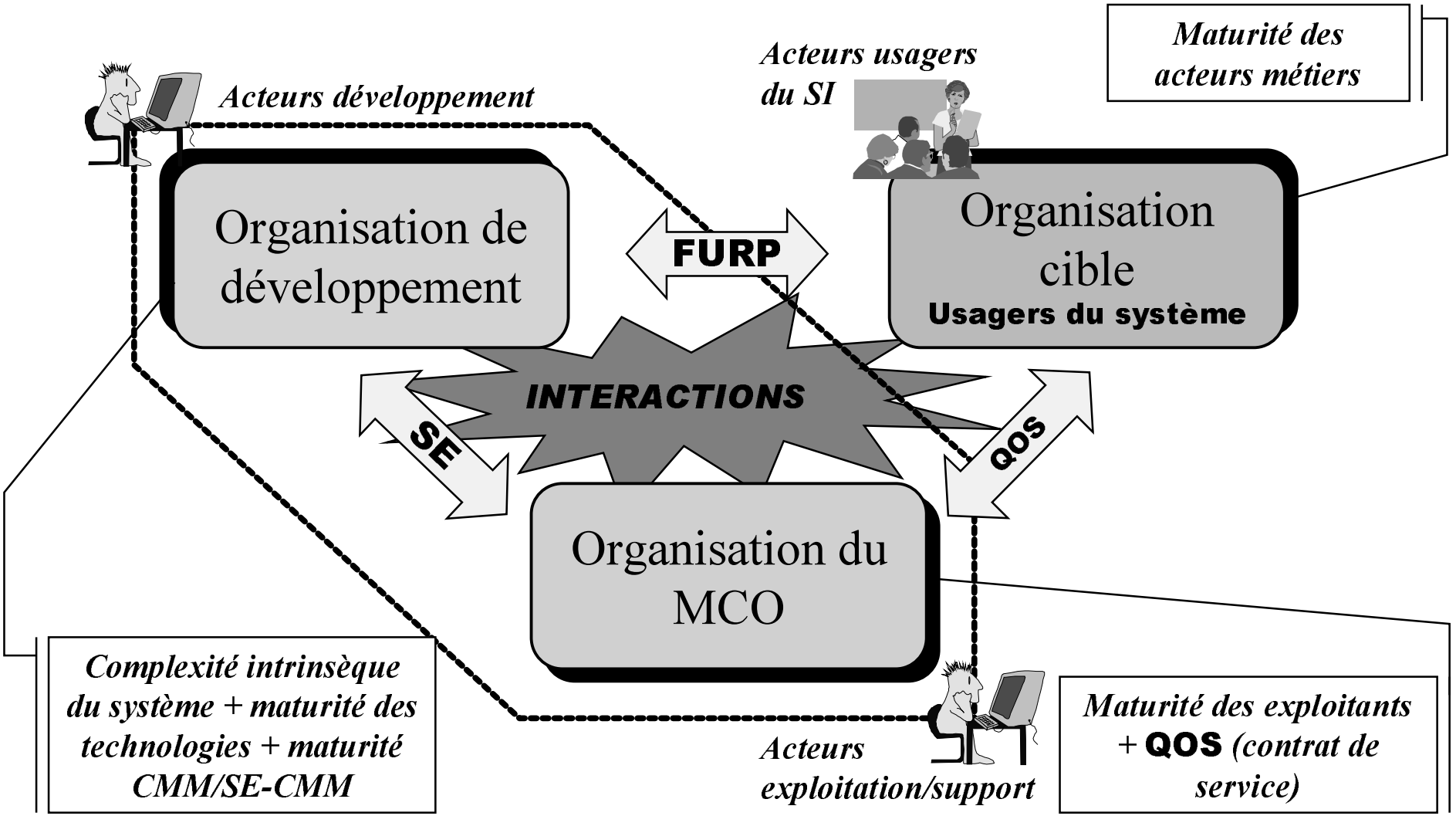
↳**COCOMO** → **CO**nstructive **CO**st **MO**del

- Identification **systématique** de toutes les tâches génératrices de **coûts objectifs**
  - ❑ Nomenclature des **éléments** du produit logiciel à réaliser et des **relations** que ces éléments ont entre eux
    - ✓ Cf. le rôle fondamental de la gestion de configuration
  - ➘ Construction – Validation de l'arbre produit
    - ✓ Spécification des interfaces - Intégration
  - ➘ Réalisation des éléments terminaux de l'arbre produit

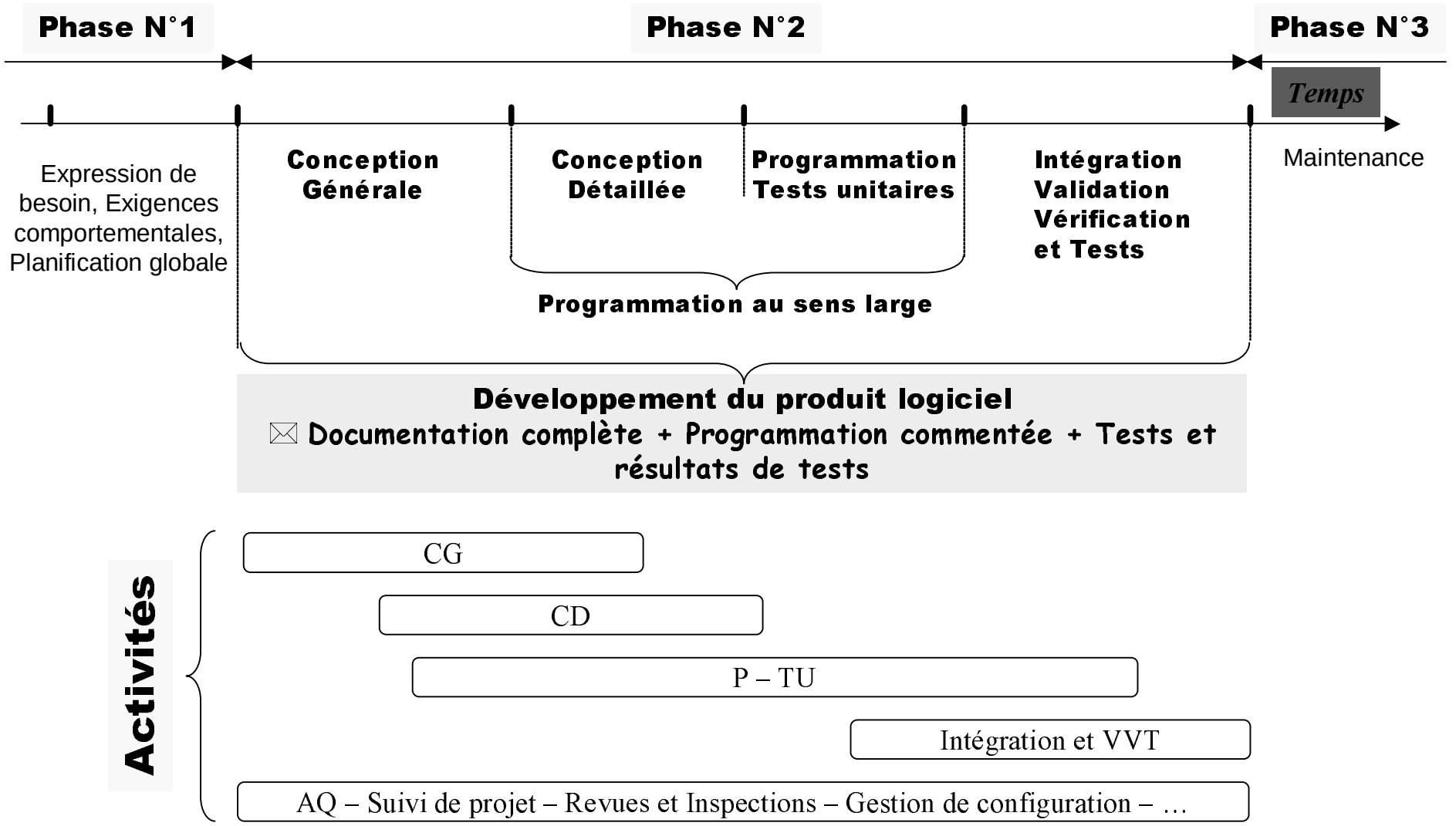
✉**Avant d'utiliser un modèle, il est impératif de comprendre son fonctionnement**

✉**Le modèle ne prendra jamais les décisions à la place du chef de projet !!!**

Les acteurs majeurs de la formation des coûts et de la complexité



COCOMO : Phases et activités (1/2)



COCOMO : Phases et activités (2/2)

Structuration en **activités**

- Approche permettant de répertorier tout ce qui doit être effectué au cours d'un projet (Organigramme des tâches – WBS)
  - **Standardisation du processus de développement**
  - Exemple : Conception générale, Conception détaillée, Programmation – Tests unitaires, Intégration, Assurance qualité globale, …

Structuration en **phases**

- Vision temporelle, faisant l'hypothèse qu'un projet est naturellement divisé en grandes étapes → i.e. les phases, effectuées les unes après les autres sans recouvrement – Jalons et dates clés
  - **Approche référentiel projet** → Identification des dates des premières versions des **livrables du projet** (appelés artefacts dans UP)
  - Chaque phase comporte **plusieurs activités menées en parallèles**

Ces visions sont orthogonales et complémentaires

Phases COCOMO

Phases et référentiel projet

- Phase 1 : Expression de besoin et Exigences comportementales (**FURPSE**) – Planification globale du projet
  - ✓ livrable : Référentiel EB/EC
- Phase 2 : **Développement**
  - ❑ Phase 2.1 : **Conception générale**
    - ✓ livrable : Référentiel **CG**
  - ❑ Phase 2.2 : Réalisation – (Programmation au sens large)
    - ↘ Phase 2.2.1 : **Conception détaillée**
      - ✓ livrable : Référentiel **CD**
    - ↘ Phase 2.2.2 : **Codage/Programmation et tests unitaires**
      - ✓ livrable : Référentiel **P/TU**
  - ❑ Phase 2.3 : **Intégration et Tests**
    - ✓ livrable : Référentiel **VVT**
- Phase 3 : Exploitation et support – Maintenance

Activités COCOMO (1/2)

- Processus de développement décomposé en **8 activités principales**
  - Expression de besoin et exigences comportementales
  - Conception Générale
    - ✓ Interfaces de haut-niveau — Prise en compte des exigences fonctionnelles et non-fonctionnelles d'un point de vue technique
  - Programmation, qui se décompose en 4 sous-activités
    - Conception Détaillée
    - Codage
    - Tests Unitaires
    - Intégration des modules
      - ✓ Tests de pré-intégration — Tout ce qui concerne les aspects VES dans le modèle de processus VEST

Activités COCOMO (2/2)

- Planification des tests
  - ✓ Définition et planification de la stratégie de test et de tout ce qui est nécessaire à la mise en œuvre de cette stratégie
- Validation et Vérification (Intégration système)
  - ✓ Spécification et exécution des tests correspondant à la conception générale (satisfaction des exigences fonctionnelles et non-fonctionnelles) et aux interfaces de haut-niveau — Chaînes de liaisons et interopérabilité — Tests de recette client
- Management de projet
  - ✓ Activité indispensable qui nécessite toujours un bon niveau d'expertise, en particulier si le chef de projet est également l'architecte du produit
- Gestion de configuration – Assurance qualité
- Documentation utilisateur
  - ✓ Rédaction des manuels pour l'organisation cible et l'exploitation

Équation générale de l'effort

↳Peut-on justifier la forme de cette équation ?

$$Effort = \underbrace{k}_{\text{Terme linéaire}} \times (KLS)^{\overbrace{1+\alpha}^{\text{Terme NON linéaire}}}$$

Facteurs d'influences

Dépend de la puissance expressive et du « grain » sémantique du langage
+ ***Expérience des acteurs***

Facteurs de coût

Dépend de la complexité de l'application et de la maturité du processus de développement
α est le ***facteur d'intégration***

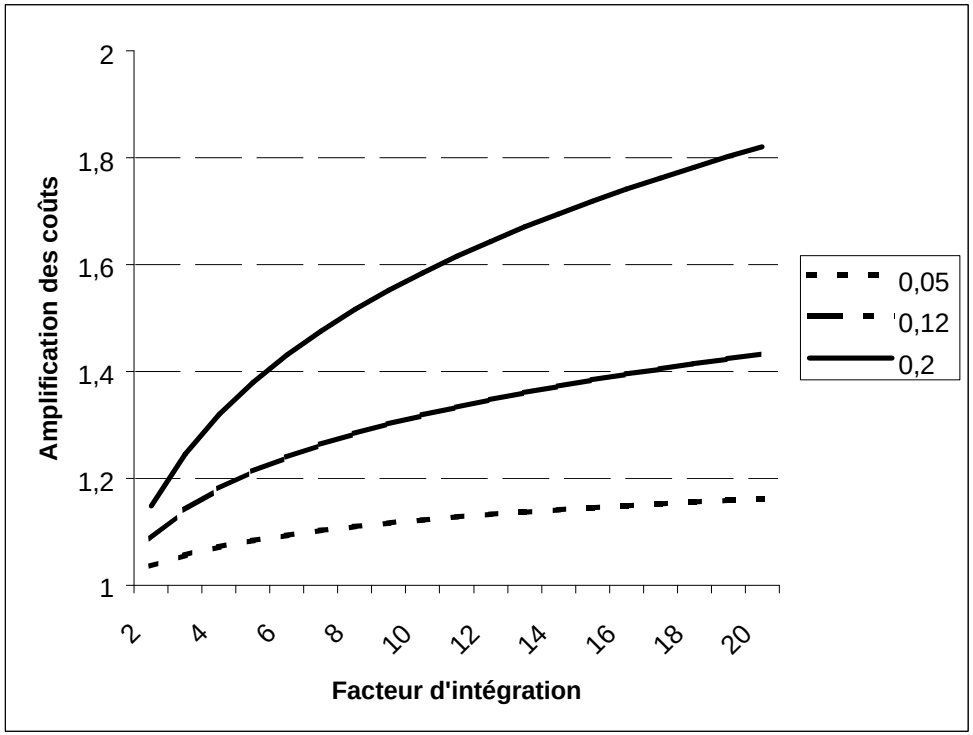
Facteurs d'échelle

Impact du facteur d'intégration α

- Tout ajout de code génère un coût d'intégration qui dépend du terme complexité

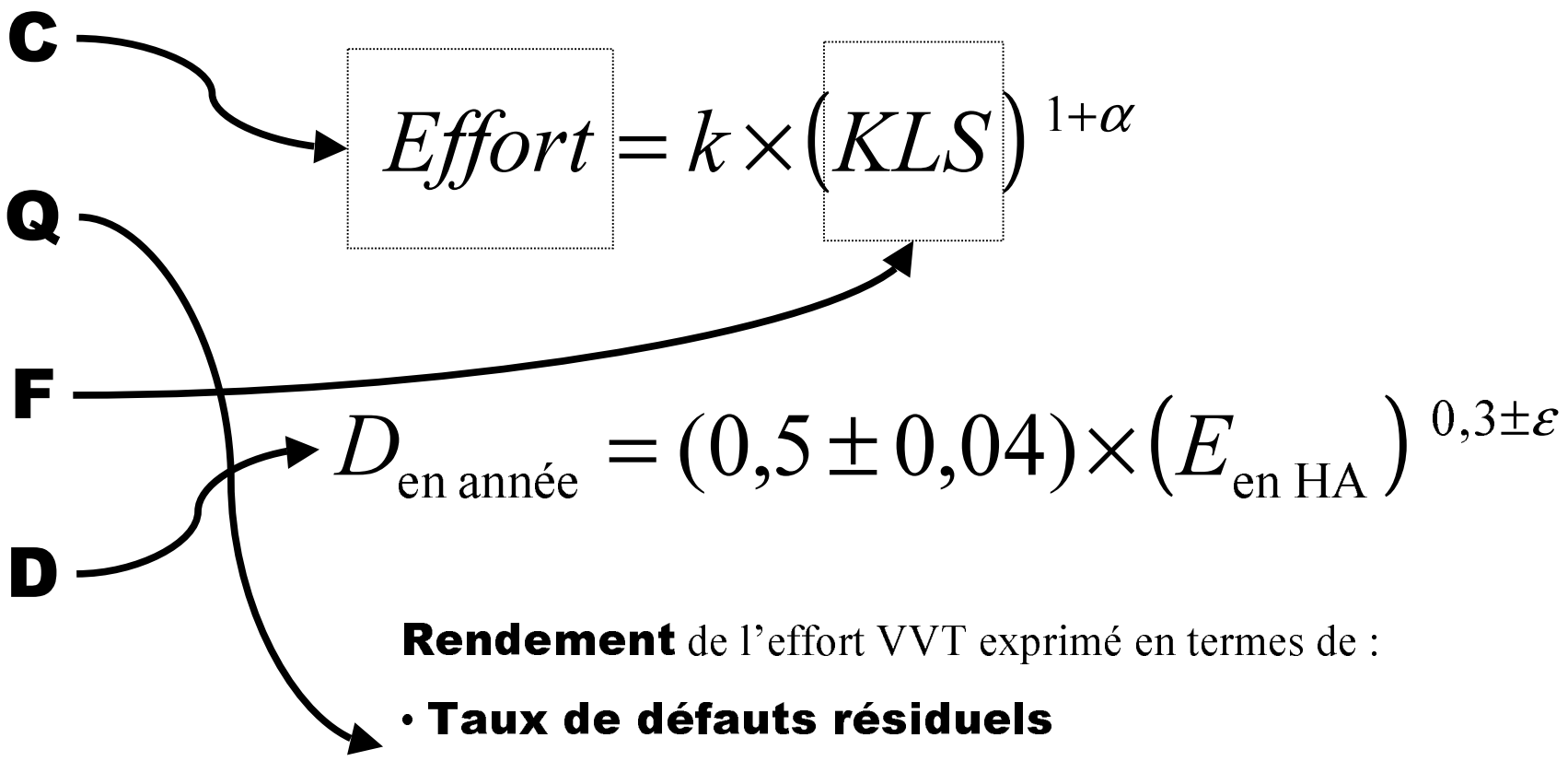
$$\xi = \frac{EFF(n \times x)}{n \times EFF(x)} = n^{\alpha}$$

Ne dépend que du nombre de modules intégrés



Courbes établies à partir des équations COCOMO basiques pour les valeur de α égales à 0,05 (S), 0,12 (P), 0,20 (E)

Interprétation des équations COCOMO avec CQFD



Rendement de l'effort VVT exprimé en termes de :

- **Taux de défauts résiduels**
- **Disponibilité de l'application ou du système**

Comptage des lignes de code (1/5)

- Définition COCOMO des Lignes de code Source
  - Taille du logiciel en **K**ilo **I**nstructions **S**ource **L**ivrées (**KISL** ou simplement **KLS**)
    - ↘ **Ce qui est réellement écrit par le programmeur et livré au client, commentaires exclus**
  - C'est une « mesure » de la **quantité de Fonctionnalités** développées pour satisfaire le besoin et les exigences exprimés
- **Problèmes** :
  - Comment compter les lignes dans la pratique ?
  - Comment normaliser l'influence des langages et des styles de programmation à ratio égal d'instructions ?

Comptage des lignes de code (2/5)

- KISL est le paramètre qui caractérise le mieux l'effort de réalisation d'un projet informatique
  - Conception : en vue de la fabrication des programmes sources livrés
  - Programmation et TU associés
    - Mesure de couverture fondée sur les graphes de contrôle associés aux instructions et sur les graphes de dépendance des données
  - Effort d'intégration IVVT fondées sur la complexité de l'architecture
    - Graphe des composants (modules) ; matrice de couplage des interfaces ; flux ; évènements ; comportements ; etc.
- Concept « d'**instruction moyenne** » résultant d'un lissage statistique et de la loi des grands nombres auquel on peut associer un quantum d'**effort moyen**
  - Unité de comptage
    - par **paquet de 1.000 ISL** (soit un effort moyen de l'ordre de 3HM ± 0,6)
      - ✓ Cf. Analogie avec la notion de quantité d'information par symbole dans la théorie de l'information de Shannon

Comptage des lignes de code (3/5)

Détermination dans la pratique

- Nécessite un solide bon sens du chef de projet
  - Très bien connaître le domaine applicatif et les styles de programmation associés
  - Maîtriser la complexité technique du projet (architecture et intégration)
  - Savoir contrôler l'environnement organisationnel et technologique
  - Savoir utiliser les composants réutilisables et les bibliothèques de composants
    - ✓ On connaît la taille de ces composants à l'avance
- Faire confiance à la loi des grands nombres et au lissage statistique
- Distinguer, si nécessaire, les lignes écrites, les lignes modifiées, les lignes supprimées
  - Ne pas oublier les lignes de JCL, Scripts de paramétrage, etc.
  - Nombreux outils disponibles pour gérer correctement les codes source
- Cela s'applique très bien à la problématique de ré-ingénierie et/ou de migration d'applications : KISL est connu par définition

Comptage des lignes de code (4/5)

Influence des langages de programmation

Classe du langage	Langage	Coefficient de puissance
Machine	Assembleur	1
Machine	Macro assembleur	1,5
LHN	C	2,5
LHN	ALGOL	3
LHN	FORTRAN	3
LHN	COBOL	3
LHN	Ada 83	4,5
LHN	Java, C++	5
LHN	Ada 95	6
LODP	Visual BASIC	10
LODP	Objective C	12
LODP	Smalltalk	15
LODP	Powerbuilder	20
LODP	SQL	25

- **Coefficient de puissance** = nombre d'instructions assembleur pour coder une instruction en LHN
- Tous les LHN ont la même puissance expressive même si certains langages sont plus concis que d'autres
  - NB : les langages à objet permettent d'éviter des duplications grâce à l'héritage ; le typage fort (Ada) évite de programmer certains contrôles

Comptage des lignes de code (5/5)

Influence des styles de programmation

- Une suite d'instructions peut s'écrire en un nombre de lignes de code source (KLCS) qui peut différer selon le style de programmation de chacun

$$KISL = KLCS \times C_N$$

- C_N, appelé **coefficient de normalisation** du source, est donc éminemment variable
  - **Ne peut être évalué que par un expert connaissant les pratiques de programmation**
- La scrutation de quelques parties de différents programmes permet le calibrage (loi des grands nombres)

Exemple : en C++

```
If (x++==a) a++ ;//une ligne
```

Équivaut à en Ada 95

```
if x=a   then    //quatre lignes
    a  := a+1 ;
    end if ;
    x  :=x+1 ;
```

Cf. les concepts de vocabulaire et de décomptage des opérateurs / opérandes de M.Halstead

Organisation du modèle

- Trois niveaux de précision
  - COCOMO **basique**
    - Typologie des programmes S / P / E + Volume de programmation
    - Ventilation de l'effort par phase et par activité de développement
  - COCOMO **intermédiaire**
    - Introduction des facteurs de coût au niveau global de l'estimation
  - COCOMO **détaillé**
    - ✓ Ce niveau requiert une **très grande expertise** en ingénierie du logiciel
    - Analyse détaillée des facteurs de coût
      - ✓ C'est en fait une check-list de risques
    - Ventilation de l'influence du facteur de coût par phase de développement

Conclusion : difficultés et limites du modèle

- **Expliciter** très clairement la **méthodologie de comptage**
  - Standardisation d'un cycle de développement de façon à imputer l'effort de développement à une nomenclature incontestable de tâches projets
    - ✓ Distinction Phases / Activités
    - ✓ Faire très attention avec les cycles de type RAD, de type UP
  - Métrologie du volume de programmation
    - ✓ Lissage des paramètres concernant le style de programmation
- **Expliciter** les hypothèses concernant les **facteurs d'influence** (complexité, facteurs de risque, incertitudes)
  - Séparer clairement le subjectif et le qualitatif, du quantitatif
    - ✓ Faire très attention aux facteurs organisationnel (MOA, MOE, Sous-traitants, etc.)
  - Révision des hypothèses dans un cadre de suivi de projet (gestion des risques)
    - ✓ Arbre de dépendance classique avec les méthodes d'analyse des données
- La pratique reproductible du modèle requiert une **véritable expertise** en management de projet logiciel

Estimation et maturité logicielle

- Sur la base du modèle de maturité CMM à 5 niveaux :
  - Au niveau 2 (le processus de développement est **défini**) : on doit maîtriser les bases de la gestion de projet
    - Modèle CQFD ; COCOMO basique et COCOMO intermédiaire
  - Au niveau 3 et au delà (le processus de développement est **reproductible** et instrumenté) : on doit maîtriser tous les aspects de l'estimation
    - COCOMO détaillé et COCOMO II
      - ✓ Requiert, au minimum, 5 ans d 'expérience et la réalisation effective de plusieurs projets de complexité variée

Le modèle COCOMO basique

Équations d'effort et de la durée

- Un jeu d'équations par type de programme et modes de développement (Unité : 1hm=152h travaillées)

Typologie des programmes – Modes de développement		
Simple de type conversion– Traitement de cas particulier Type ***S*** – (Organic)	***Algorithmique–Résolution d'un Problème général*** Type ***P*** – (Semi-detached)	***Enfoui–Réactif*** de type temps réel synchrone ou asynchrone Type ***E*** – (Embedded)
$Effort_{HM} = 2.4 \times KLS^{1.05}$ $D_{mois} = 3.5 \times KLS^{0.40}$	$Effort_{HM} = 3.0 \times KLS^{1.12}$ $D_{mois} = 3.7 \times KLS^{0.39}$	$Effort_{HM} = 3.6 \times KLS^{1.20}$ $D_{mois} = 3.8 \times KLS^{0.38}$
$D_{année} = 0.54 \times (E_{HA})^{0..38}$	$D_{année} = 0.50 \times (E_{HA})^{0..35}$	$D_{année} = 0.46 \times (E_{HA})^{0..32}$
L'approximation $D_{année} = 0.5 \times \sqrt[3]{Effort_{HA}}$ est légitime		

Analyse de la typologie S P E

- La typologie S P E correspond à des **niveaux de complexité** tant au niveau de l'architecture **produit** que de l'**organisation** du projet de développement
- 3 niveaux de complexité
  - ***S*** – Linéaire simple : $k \times N$
    [semi-]logarithmique : $k \times \log(N)$ ou $k \times N \times \log(N)$
  - ***P*** – Polynomial : $k \times (N)^{\alpha}$ *avec* $\alpha = 2$ *ou* 3
  - ***E*** – Exponentiel : $k \times (\alpha)^{N}$ et au dela $k \times (\alpha^{N})^{M}$ …

Allure des courbes et approximations (1/3)

- Les courbes sont polynomiales :

$$y = k \times (x)^{1+\alpha} \quad ; \text{dérivée} \rightarrow y' = k(1+\alpha) \times x^{\alpha}$$

Pente de la droite d'approximation linéaire pour x=1

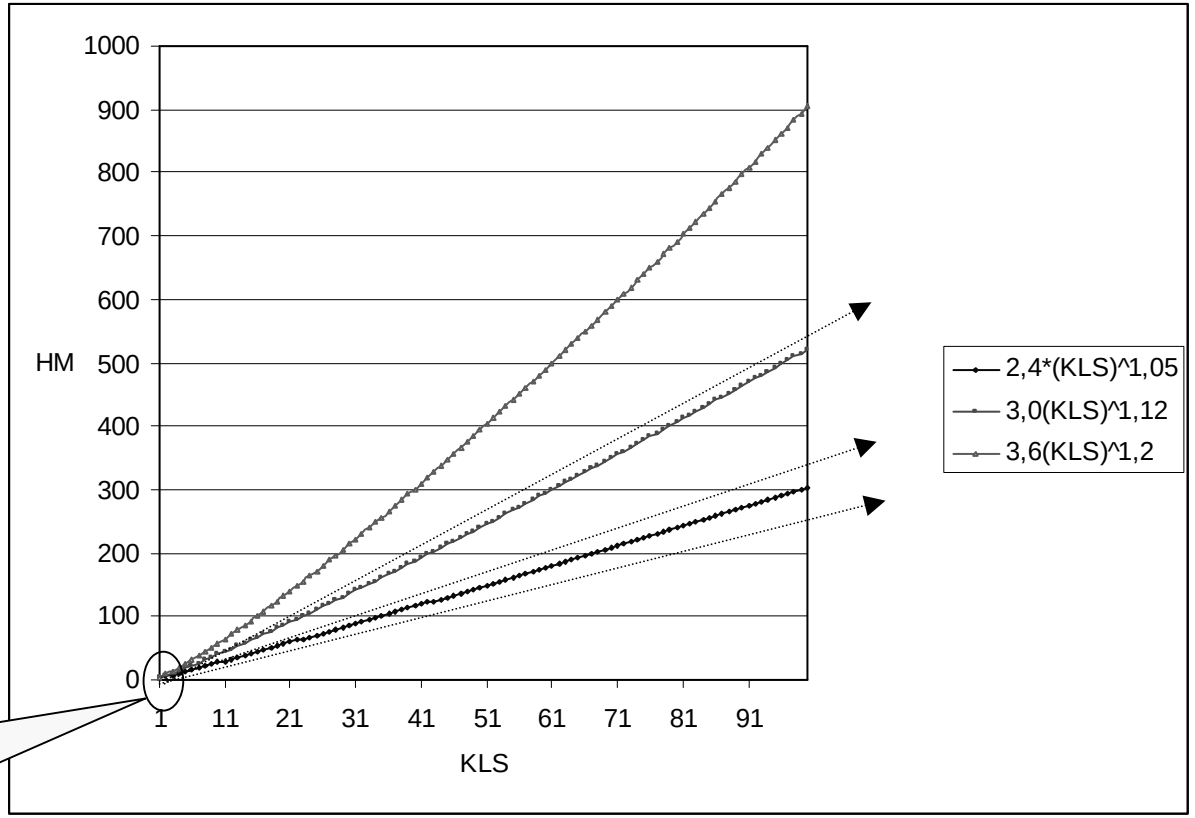
- Approximations linéaires :

$$y = 2{,}4 \times (x)^{1{,}05} \quad ; \text{Approximation linéaire} \rightarrow y_L = 2{,}4 + 2{,}52 \times (x-1)$$

$$y = 3{,}0 \times (x)^{1{,}12} \quad ; \text{Approximation linéaire} \rightarrow y_L = 3{,}0 + 3{,}36 \times (x-1)$$

$$y = 3{,}6 \times (x)^{1{,}20} \quad ; \text{Approximation linéaire} \rightarrow y_L = 3{,}6 + 4{,}32 \times (x-1)$$

Allure des courbes et approximations (2/3)



Valeurs à l'origine :
- **2,4 hm**
- **3,0 hm**
- **3,6 hm**

Allure des courbes et approximations (3/3)

Exemples de marges d'erreurs

➘ Cas $x = 100\ \text{KLS}$

- Type S

$$y = 2{,}4 \times (100)^{1{,}05} = 302\,;\ \text{Approximation linéaire} \rightarrow y_L \cong 252$$

$$\Delta = \frac{302 - 252}{252} \cong 20\%$$

- Type P

$$y = 3{,}0 \times (100)^{1{,}12} = 521\,;\ \text{Approximation linéaire} \rightarrow y_L \cong 336$$

$$\Delta = \frac{521 - 336}{336} \cong 55\%$$

- Type E

$$y = 3{,}6 \times (100)^{1{,}20} = 904\,;\ \text{Approximation linéaire} \rightarrow y_L \cong 431$$

$$\Delta = \frac{904 - 431}{431} \cong 110\%$$

Exemples de complexités fréquentes

- Une opération qui requiert un tri de N entités est en
$$O(k \times N \times \log(N))$$
- Une opération qui requiert de comparer ou de communiquer entre N entités est en :
$$O(N^2)$$
- Une opération qui requiert de manipuler l'ensemble des parties de N entités est en :
$$O(2^N)$$
- Une opération qui requiert de manipuler un ordre de N entités est en :
$$O(N!) \approx O\left(\left(\frac{N}{e}\right)^N \sqrt{2\pi \times N}\right)$$

Type S

- Programme **transformationnel** bien spécifié qui ne requiert qu'une compétence générale en informatique (→ connaissance d'un langage de programmation) – Organisation de projet simple ne nécessitant que très peu d'interactions (→ transactions simples sur une base de données relationnel) – Aucune innovation
  - Migration de code iso-fonctionnel
  - Exemple de programme de ce type : calcul de $\sqrt{A}$ Algorithme connu de puis longtemps (suite de Newton)

$$\sqrt{A} = \underset{n\to\infty}{Lim}\left(x_{n+1} = \frac{1}{2}\left(x_n + \frac{A}{x_n} \right) \right)$$

Type *P*

- Programme **transformationnel** contenant des **algorithmes** (avec des contraintes de performance) qui requiert une grande compétence (→ connaissance approfondie d'une ou plusieurs bibliothèques de services ; architecture en couches/clients-serveurs avec interfaces d'accès, etc.) — Organisation de projet qui nécessite des **interactions nombreuses** principalement entre les membres de l'équipe projet – Présence d'innovations significatives
  - Exemple : applications de Data-Mining et/ou très grande base de données, EAI nécessitant la réalisation de nombreux connecteurs (ce sont des traducteurs-convertisseurs d'interfaces) de façon à faire inter-opérer des systèmes initialement indépendants

Type *E*

- Programme **réactif** en environnement système instable contenant des automates de contrôles à créer (avec des contraintes de sûreté et de haute disponibilité, gestion et partage de ressources, gestion des priorités, etc.) qui requiert une très grande expertise (→ connaissance approfondie du contexte système) – Organisation de projet complexe qui nécessite des interactions nombreuses avec l'**organisation cible** et les **exploitants** ainsi que le respect de normes externes (→ certification de composants critiques, etc.) – Forte innovation
  - Systèmes temps réel, Systèmes de contrôle-commande « intelligent », couches basses de protocoles et infrastructures, drivers de périphériques, IHM avec navigation complexe, etc.
    - ✓ Spécification rigoureuse des interfaces et maîtrise de la combinatoire ; Validation à l'aide de simulateurs d'environnement ; Gestion de configuration impérative pour manager les modifications inévitables dans ce contexte, etc.

Facteurs de complexité – Influence des exigences non fonctionnelles (1/3)

- Multilinguisme
  - Longueur des textes très variable selon la langue (exp. → 30% de plus en français qu'en anglais !!!)
    - ✓ Gestion mémoire très complexe — Abstractions linguistique pour le stockage
- Traitement d'erreurs « intelligent »
  - Localisation exacte des défauts suite au constat d'une défaillance – Correction guidée et/ou automatique
    - ✓ Gestion de traces contextuelles de longueur quelconque — Points de reprise prédéfinis
- IHM « intelligent » multi-acteurs (novice, expert, instrumentation/audit, administrateur)
  - Nécessité de connaître les modèles comportementaux des acteurs – Langages de commandes cachés reconstruit dynamiquement
    - ✓ Automates complexes à grand nombre d'états — Taille des automates — Performance du point de vue de l'usager (temps de réponse, etc.)

Facteurs de complexité – Influence des exigences non fonctionnelles (2/3)

- Tolérance à certains types de pannes – Modes dégradés – Dispositifs d'auto-contrôle et auto-réparation
  - Construction d'un modèle de pannes et tables d'exceptions fonctionnelles prises en compte par l'applicatif – Variété des réponses
    - ✓ Cette partie de l'application est totalement dépendantes de l'architecture et du style de programmation adopté (cf. programmation par contrat → quand le contrat est violé, que fait-on ???) — Présence de non-déterminisme si stratégies de réparation !!! — Gestion de ressources très complexe
- Paramétrage – Auto-adaptation et/ou configuration automatique selon contexte d'installation et/ou de l'environnement
  - Identification des types abstraits de données – Abstractions sémantiques et maîtrise des modèles et méta-modèles associés
    - ✓ Architectures très complexes qui requièrent une connaissance approfondie des techniques de compilation, des modèles de données et de leurs diverses représentations — Notions d'équipements virtuels compilables dynamiquement — Gestion de ressources très complexes

Facteurs de complexité – Influence des exigences non fonctionnelles (3/3)

- Évolutivité – Extensions fonctionnelles avec garantie de compatibilité ascendante – Extensions du modèle de données (paramétrage et méta-description)
  - Programmation à l'aide de tables descriptives des éléments susceptibles d'évoluer (automates de contrôle et/ou données)
    - ✓ Interprétation et/ou compilation dynamique des modèles sous-jacents (cf. cas des moteurs relationnels, du traitement des E/S dans les drivers par les programmes canaux) — Gestion de caches complexes pour pallier les problèmes de performance inhérents à ces exigences
- Migration d'applications entraînant une forte augmentation de complexité
  - Migration Batch → Conversationnel (OLTP + Messages) – Migration OLTP centralisé → OLTP distribué n-tiers + Transactions longues
    - ✓ Validation des nouvelles chaînes de liaisons ainsi créées - Traitement des erreurs systèmes et relations avec les JCL — Non déterminisme créé par les protocoles supportant les échanges et contrôle de l'état global — Caches répartis sur *n* machines — Traitement des reprises en environnement distribué

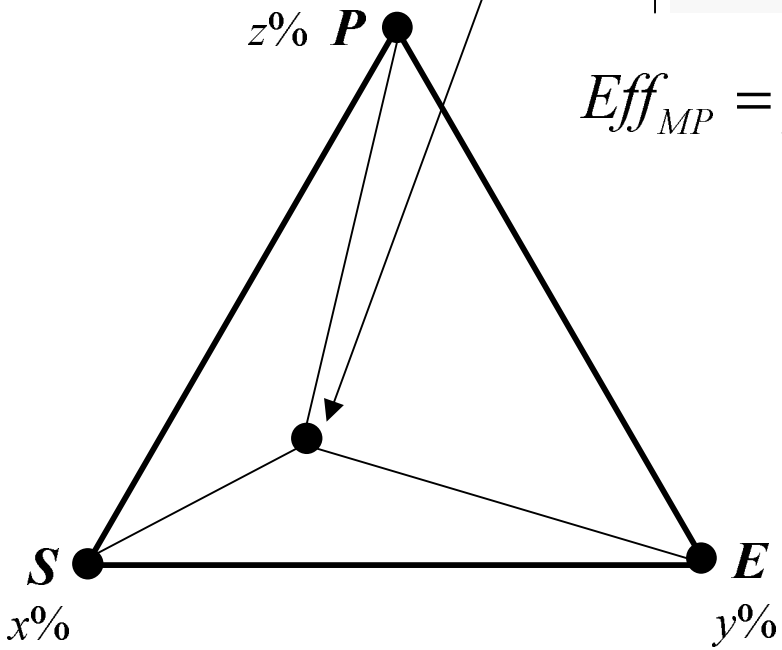
Exemple d'un gestionnaire de ressources

- La caractérisation de l'usage des ressources est un facteur de complexité très important
  - ✓ **Classification** de l'usage des ressources (Computing Survey N°27/Vol. 3/95)
  - ❑ Homogène / Hétérogène
    - ✓ Des ressources homogènes sont identiques. Une demande de ressource n'a pas besoin de spécifier le type de la ressource. Dans le cas contraire, elles sont hétérogènes. Il faut manipuler plus d'information pour les utiliser (exp. : différentes catégories de mémoire → persistante/non persistante, de fichiers et méthodes d'accès, de protocoles de communications, etc.).
  - ❑ Spécifique / Général
    - ✓ Une ressource est spécifique si elle correspond à une requête particulière qui ne peut s'exécuter que si la ressource en question est disponible. Dans le cas contraire la demande est dite générale (exp. : traitements des exceptions spécifiques à un contexte, ou non spécifiques).
  - ❑ Temporaire / Durable
    - ✓ Une demande de ressource est durable si elle perdure au delà de la requête qui l'a initialement demandée (Exp. : une ouverture de fichier, après une lecture) ; elle doit être explicitement libérée en fin d'usage (cf. ACID).
  - ❑ Régulier / Irrégulier
    - ✓ Implique la persistance, mais les conflits éventuels sont différés à la fin de l'exécution de la fonction (Exp.: écritures différées en fin de transaction)

Programmes composites

- La plupart des programmes réels sont des mélanges S, P et E : comment calculer l'effort pour *N* KLS ?

$$N_{KLS} = x_S \times N_{KLS} + y_P \times N_{KLS} + z_E \times N_{KLS} \; avec \; x + y + z = 1$$



Effort moyen pondéré (barycentre)

$$Eff_{MP} = \underline{x \times 2{,}4(N)^{1{,}05}} + y \times 3{,}0(N)^{1{,}12} + z \times 3{,}6(N)^{1{,}20}$$

A comparer avec l'effort pour développer N_S seul

$$Eff_{N_S} = 2{,}4(x \times N)^{1{,}05} = \underline{x^{1{,}05}} \times 2{,}4(N)^{1{,}05}$$

$$NB : x > x^{1{,}05} \; car \; x < 1$$

Distribution de l'effort par phase

Répartition de l'effort (en % de l'effort de développement) Typologie	Phase	32 KISL	128 KISL	512 KISL
S	Expression de besoin et planification	6	6	
	Conception générale	16	16	
	Réalisation	62	59	
	Conception détaillée	24	23	
	Codage et tests unitaires	38	36	
	Tests et intégration	22	25	
P	Expression de besoin et planification	7	7	7
	Conception générale	17	17	17
	Réalisation	58	55	52
	Conception détaillée	25	24	23
	Codage et tests unitaires	33	31	29
	Tests et intégration	25	28	31
E	Expression de besoin et planification	8	8	8
	Conception générale	18	18	18
	Réalisation	54	51	48
	Conception détaillée	26	25	24
	Codage et tests unitaires	28	26	24
	Tests et intégration	28	31	34

Σ=100

Σ=106

Distribution de la durée par phase

Répartition de la durée (en % de la durée de développement) Typologie	Phase	32 KISL	128 KISL	512 KISL
S	Expression de besoin et planification	12	13	
	Conception générale	19	19	
	Réalisation	55	51	
	Tests et intégration	26	30	
P				
	Expression de besoin et planification	20	22	24
	Conception générale	26	27	28
	Réalisation	48	44	40
	Tests et intégration	26	29	32
E				
	Expression de besoin et planification	32	36	40
	Conception générale	34	36	38
	Réalisation	40	36	32
	Tests et intégration	26	28	30

Correspondance Phases-Activités

Exemple pour des programmes de type **S**

Phase	Expression de besoin et planification		Conception générale		Réalisation		Tests et intégration	
Taille du produit (KISL)	32	128	32	128	32	128	32	128
Répartition de l'effort par phase (%)	6		16		62	59	22	25
Pourcentage d'activité par phase								
Analyse des besoins	46		15		5		3	
Conception générale	20		40		10		6	
Conception détaillée	3		14		58		34	
Planification des tests	3		5		4		2	
Vérification et validation	6		6		6		34	
Conduite de projet	15		11		6		7	
Gestion de configuration et assurance qualité	2		2		6		7	
Documentation	5		7		5		7	

Σ=100

Poids des activités dans la phase

Relations quantitatives entre les activités

- Sur la base des activités du cycle de développement

  (cf. COCOMO et/ou équivalent ISO12207) :

  - **EB** Expression de besoin, **CG** Conception générale, **DEV** Développement **projet** (Conception détaillée + Programmation Tests unitaires + VVT), Logistique **projet** (Management de projet, Gestion de configuration, Assurance Qualité, Documentation livrée)

- On a les relations :

  - **Effort Total** $\approx$ **20** $\times$ [Effort **EB**]
    - ✓ 17 pour S ; 20 pour P ; 25 pour E
  - **Effort projet** $\approx$ **7** à **8** $\times$ [Effort **CG**]
    - ✓ 6,7 pour S ; 7,3 pour P ; 8,0 pour E
  - **Effort DEV** $\approx$ **5** $\pm$ **0,5** $\times$ [Effort **CG**]
    - ✓ 4,43 pour S ; 4,62 pour P ; 5,50 pour E

Relations quantitatives entre les phases

- Sur la base des phases du cycle de développement COCOMO :
  - ☹ à manipuler avec grande précaution à cause des recouvrements entre les phases
- On a les relations :
  - **Effort projet** ≈ **6** × [Effort phase **CG**]
    - ✓ S = 6,3 ; P = 5,9 ; E = 5,6
  - **Effort P/TU** ≈ **3** × [Effort phase **CG**]
    - ✓ S = 3,7 ; P = 3,2 ; E = 2,8
  - **Effort VVT** ≈ **1,5** × [Effort phase **CG**]
    - ✓ S = 1,56 ; P = 1,64 ; E = 1,72

Le modèle COCOMO intermédiaire

Les facteurs de coût du modèle
version 81

Influence des facteurs de coûts

- Les facteurs de coût permettent de calibrer plus précisément le coefficient k de l'équation d'effort
- Équation d'effort du modèle COCOMO intermédiaire :
  - ***S*** $Eff_{Nominal} = 3{,}2 \times k \times (KLS)^{1{,}05}$
  - ***P*** $Eff_{Nominal} = 3{,}0 \times k \times (KLS)^{1{,}12}$
  - ***E*** $Eff_{Nominal} = 2{,}8 \times k \times (KLS)^{1{,}20}$

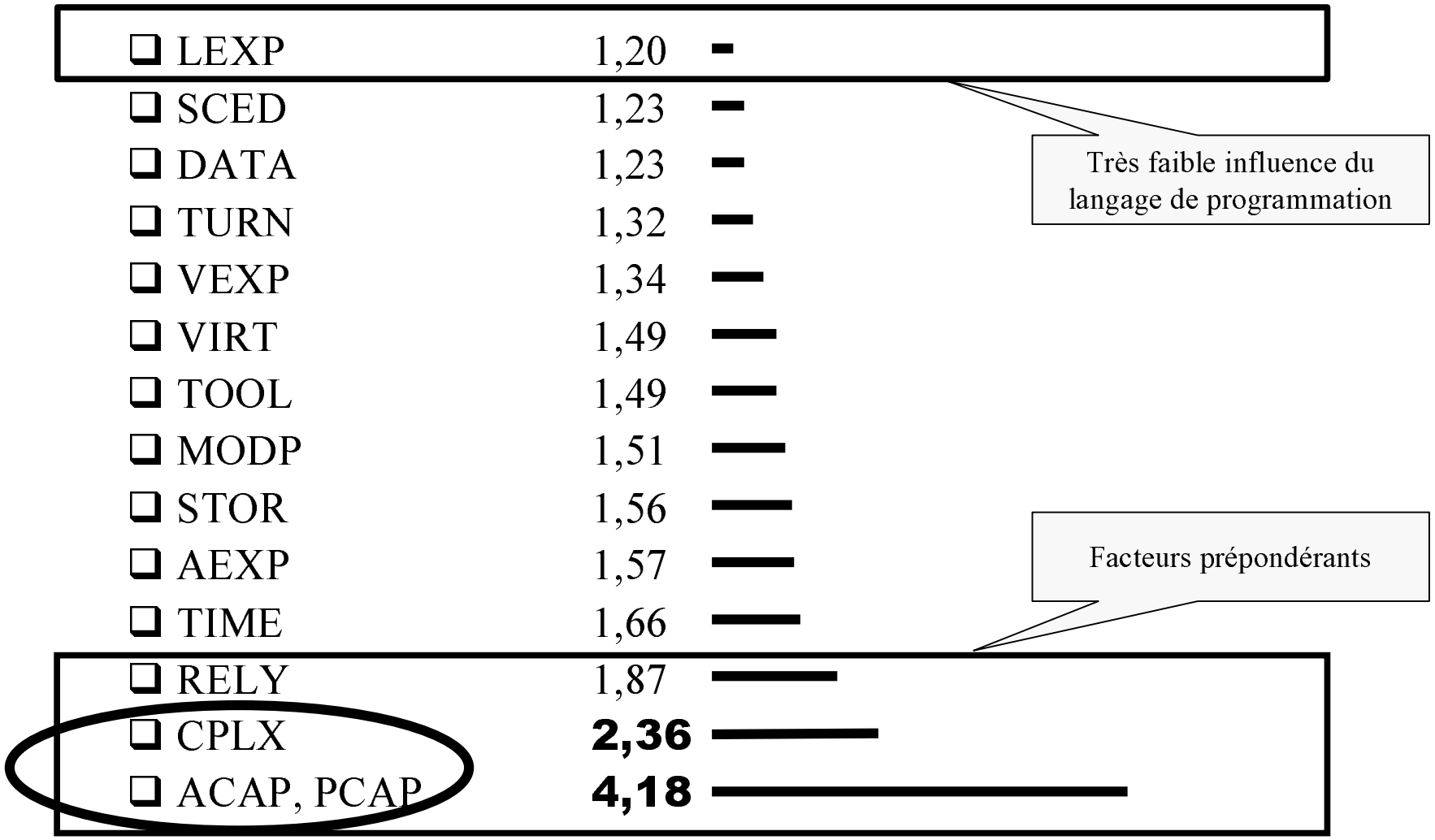
Table des facteurs de coût

	Niveaux					
Facteurs de Coûts	Très bas	Bas	Nominal	Élevé	Très élevé	Extrêmement élevé
Attributs du produit (Complexité de l'application et/ou du produit à réaliser						
RELY Contraintes de fiabilité du logiciel	.75	.88	1.00	1.15	1.40	N/A
DATA Taille de la base de donnée	N/A	.94	1.00	1.08	1.16	N/A
CPLX Complexité de l'application	.70	.85	1.00	1.15	1.30	1.65
Attributs du matériel (Qualité de service du centre de calcul et de l'ordinateur cible)						
TIME Contraintes de la durée d'exécution sur le temps machine	N/A	N/A	1.00	1.11	1.30	1.66
STOR Contraintes d'occupation mémoire	N/A	N/A	1.00	1.06	1.21	1.56
VIRT Stabilité de la machine virtuelle	N/A	.87	1.00	1.15	1.30	N/A
TURN Durée d'exécution	N/A	.87	1.00	1.07	1.15	N/A
Attributs du personnel (Expérience et/ou maturité des programmeurs, individuellement et collectivement)						
ACAP Maturité des architectes	1.46	1.19	1.00	.86	.71	N/A
AEXP Expérience du domaine applicatif	1.29	1.13	1.00	.91	.82	N/A
PCAP Maturité des programmeurs	1.42	1.17	1.00	.86	.70	N/A
VEXP Expérience d'utilisation de l'environnement d'exploitation	1.21	1.10	1.00	.90	N/A	N/A
LEXP Pratique des langages de programmation utilisés	1.14	1.07	1.00	.95	N/A	N/A
Attributs du projet (Processus et/ou méthodes de développement adoptés par le projet)						
MODP utilisation des pratiques modernes de programmation	1.24	1.10	1.00	.91	.82	N/A
TOOL Utilisation d'outils logiciels	1.24	1.10	1.00	.91	.83	N/A
SCED Mise en place d'un planning de réalisation	1.23	1.08	1.00	1.04	1.10	N/A

Exemple d'évaluation du facteur k

Facteurs de coût	Plage de valeurs	Exemple de valeur	Remarques / caractéristiques
Produit			
RELY	0,75 ; 1,4	1,15	L'application a des contraintes de fiabilité importantes
DATA	0,94 ; 1,16	0,94	Pas de base de données.
CPLX	0,7 ; 1,65	1,15	Les algorithmes à utiliser sont complexes.
Ordinateur cible			
TIME	1 ; 1,66	1,11	L'exécution du programme occupe 70% du temps machine
STOR	1 ; 1,56	1,00	L'exécution du programme demande moins de 50% de ressource mémoire
VIRT	0,87 ; 1,3	1,00	L'architecture de la machine cible est classique.
TURN	0,87 ; 1,15	1,00	La durée d'exécution est inférieure à 4 heures.
Expérience du personnel			
ACAP	0,71 ; 1,46	0,86	L'architecte est bon.
AEXP	0,82 ; 1,29	1,13	Il a peu d'expérience de ce type d'application.
PCAP	0,7 ; 1,42	1,17	Les programmeurs sont débutants.
VEXP	0,9 ; 1,21	0,90	Les programmeurs connaissent l'environnement d'exploitation.
LEXP	0,95 ; 1,14	1,07	Le langage de programmation est nouveau pour les programmeurs.
Projet			
MODP	0,82 ; 1,24	0,91	Les programmeurs connaissent les méthodes modernes de programmation.
TOOL	0,83 ; 1,24	0,91	Les programmeurs ont une bonne expérience d'utilisation d'outils de développement.
SCED	1 ; 1,23	1,10	Le planning de réalisation est très tendu.
Produit des valeurs obtenues : *k*		**1,376**	

Importance relative des facteurs de coût



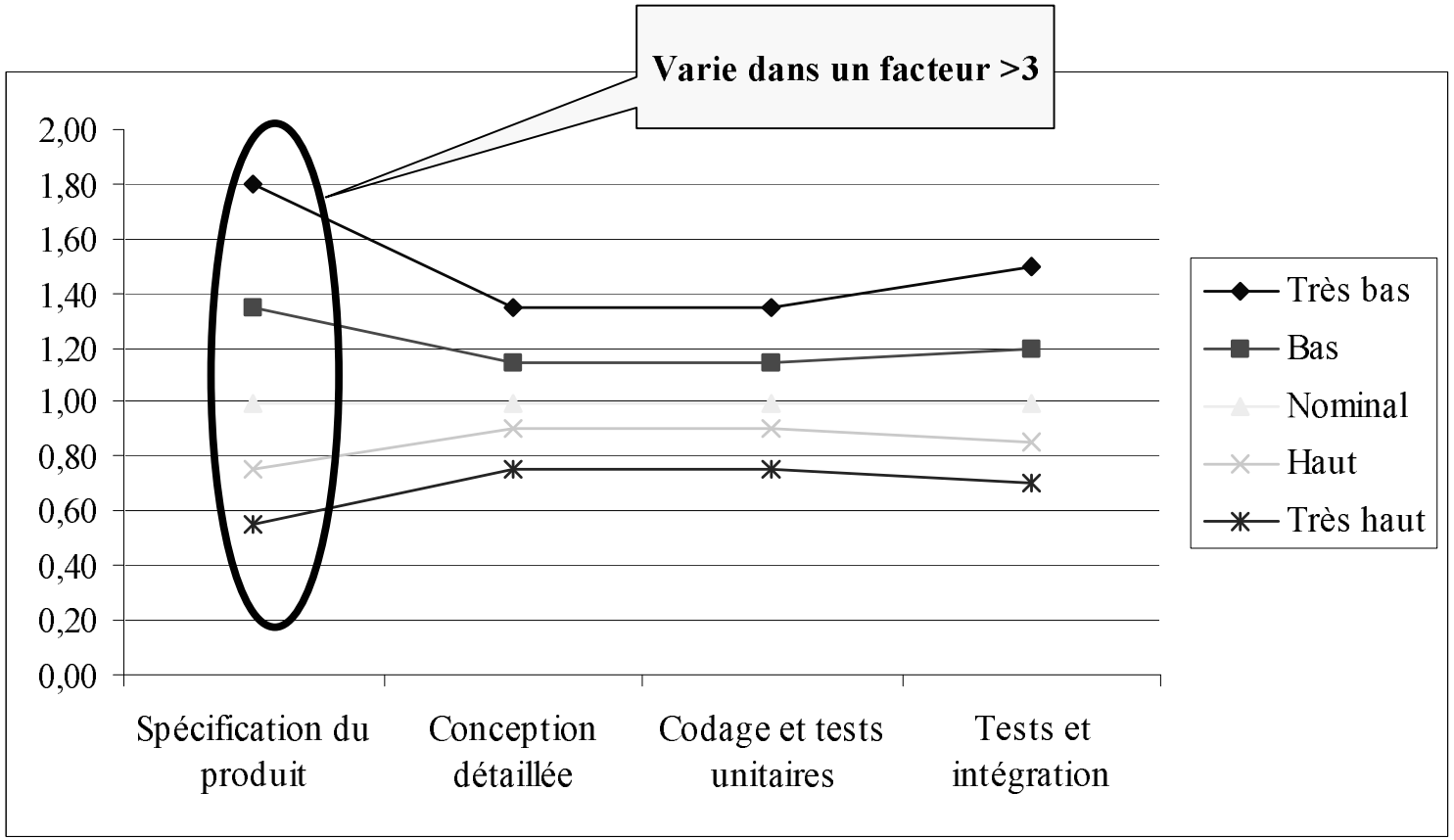
Modèle COCOMO détaillé
Apports de la version COCOMO II

Principes du modèle détaillé

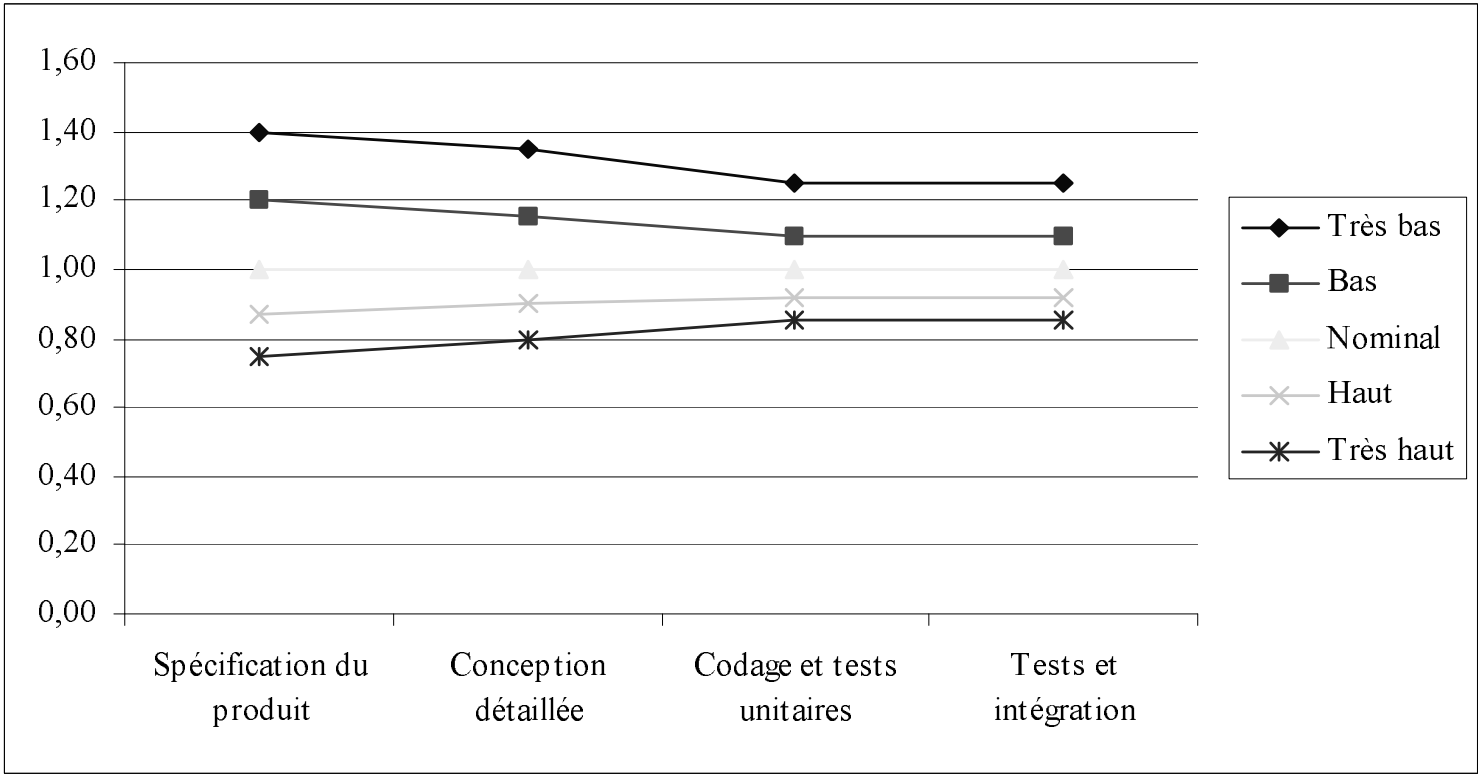
- Le modèle COCOMO détaillé propose un ensemble de critères permettant d'ajuster plus finement les facteurs d'influence en fonction des phases et des activités
  - ✓ Il est intuitivement évident que l'expérience de l'architecte — facteur ACAP —est très importante en phase de Conception (l'architecte ne code pas, ou très peu !!!)
  - ✓ Dans un cycle système (ou un cycle itératif), l'architecture doit être stabilisée dès le début
- **L'utilisation correcte du modèle détaillé et de COCOMO II nécessite un haut niveau d'expertise**

Détail du facteur ACAP par phase

- Le facteur est très important pour la spécification de l'architecture



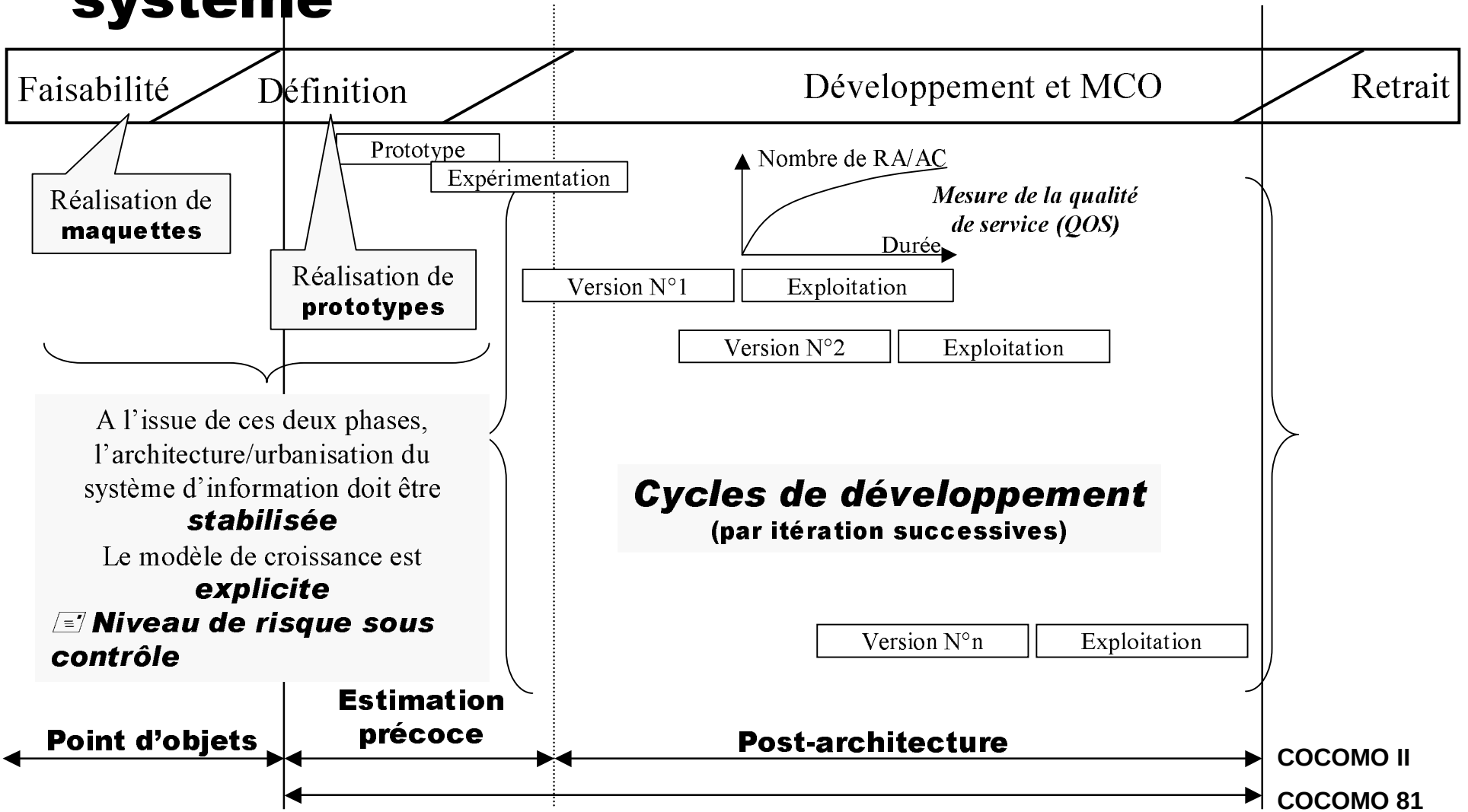
Détail du facteur AEXP par phase



Positionnement du Modèle COCOMO II

(1/2)

- S'appuie sur un cycle de vie de type **ingénierie système**



Positionnement du Modèle COCOMO II

(2/2)

- A chaque étape du cycle correspond un modèle
  - FAISABILITE : point d'objets → détermination du périmètre fonctionnel
    - Même philosophie que le modèle des PF : évaluation du nombre d'objets manipulables par l'application
      - ✓ Permet de déterminer une première valeur de l'effort
  - DEFINITION : modèle d'estimation précoce
    - Est basé sur une équation de l'effort de même type que COCOMO 81
    - **7 facteurs de coût** agrégés, **5 facteurs d'échelle** à la place de la typologie S, P et E — Intègre la problématique maturité (CMM)
  - DEVELOPPEMENT : modèle post-architecture
    - Dans la continuité du modèle précédent
    - Granularité plus fine que le modèle d'estimation précoce
      - ✓ **17 facteurs de coût détaillés** (au lieu de 15)**, les mêmes 5 facteurs d 'échelle**
  - MAINTENANCE

Équation d'effort COCOMO II

↳Forme de l'équation : $Effort_{HM} = k \times (KISL)^{1+\alpha}$

$$k = cte \times \prod Facteurs\_de\_coût \qquad cte = 2.94$$

$$\alpha = -0.09 + 0.01 \times \sum Facteurs\_d'échelle$$

- 5 facteurs d'échelle se substituent aux caractéristiques S, P, E et forment un continuum à l'appréciation du chef de projet
  - ↘ Économie d'échelle lorsque $\alpha<0$

Correspondance des facteurs de coût

Facteur de Coût COCOMO 81	Facteur de Coût COCOMO 2000		Remarques
	Estimation Précoce	Post Architecture	
RELY	RCPX	RELY	Identique
DATA		DATA	Identique
CPLX		CPLX	Identique ; mais ajout d'un domaine IHM
TIME	PDIF	TIME	Identique
STOR		STOR	Identique
VIRT		PVOL	Identique
TURN	—		Sans objet pour COCOMO 2000
ACAP	PERS	ACAP	Identique
PCAP		PCAP	Identique
AEXP	PREX	APEX	1 an d'expérience est coté L dans 81 et N en 2000
VEXP		PLEX	Identique
LEXP		LTEX	Identique
TOOL	FCIL	TOOL	Les outils ont été significativement améliorés. Ce qui était nominal en 81 est VL en 2000.
MODP		***PMAT***	La maturité du processus remplace MODP ; **PMAT est un facteur d'échelle**
SCED	SCED	SCED	Identique
	RUSE	RUSE	Nouveau : Réutilisation composants de l'application
	RCPX	DOCU	Nouveau : Besoins de documentation couverts
	PERS	PCON	Nouveau : Turnover du personnel
	FCIL	SITE	Nouveau : Développement sur plusieurs sites

Facteurs d'échelle COCOMO II

Facteurs d'échelle du modèle COCOMO 2000

- Plage de variation de $1+\alpha$
  - **De 0.91 à 1.2262**
    - ✓ Au lieu de 1,05 à 1,20 pour *COCOMO* 81
  - Quand $\alpha<0 \rightarrow$ **économie d'échelle**
    - ✓ Très forte maturité de l'équipe et de l'organisation MOE
    - ✓ Très bonne maîtrise des architectures
  - Le nominal correspond à $\alpha = 0,09$

Facteurs d'échelle	Très bas	Bas	Nominal	Elevé	Très élevé	Extrêmement élevé (0)
Précédence	Totalement sans précédent : 6.20	En grande partie sans précédent : 4.96	En partie sans précédent : 3.72	Plutôt familier : 2.48	En grande partie familier : 1.24	Totalement familier
Flexibilité du développement	Figé : 5.07	Flexibilité occasionnelle : 4.05	Une certaine flexibilité : 3.04	En conformité en général : 2.03	D'une certaine conformité : 1.01	Objectifs principaux
Architecture / résolution des risques	Un peu (20%) : 7.07	En partie (40%) : 5.65	Souvent (60%) : 4.24	Généralement (75%) : 2.83	La plupart du temps (90%) : 1.41	Toujours (100%)
Cohésion de l'équipe	Interactions très difficiles : 5.48	Interactions plutôt difficiles : 4.38	Coopération de base : 3.29	Coopération appréciable : 2.19	Coopération importante : 1.10	Coopération sans faille
Maturité des processus	Niveau 1 bas : 7.8	Niveau 1 haut : 6.24	Niveau 2 : 4.68	Niveau 3 : 3.12	Niveau 4 : 1.56	Niveau 5