Ingénierie Logicielle

Mesure et Qualité des logiciels

Prof. Ousmane SALL, Maître de Conférences CAMES en Informatique

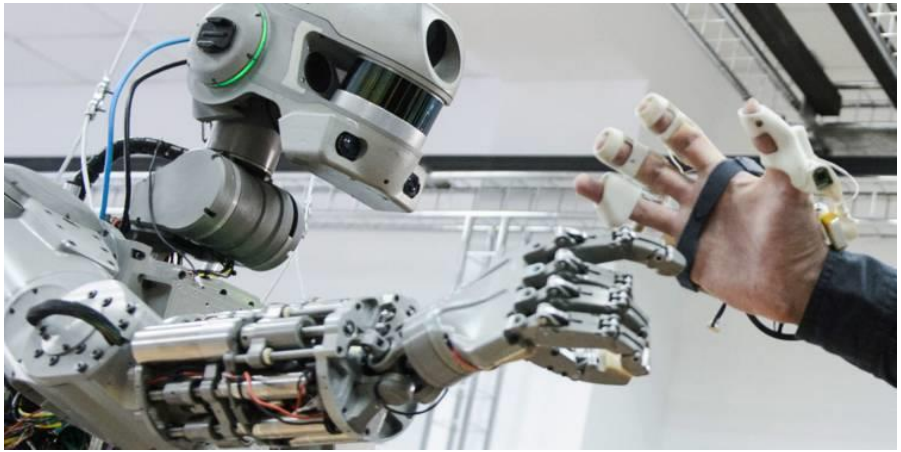
Objectifs de la séquence

- A l'issue de cette séquence vous serez capable de :
  - Décrire les critères et méthodes de mesure d'un logiciel;
  - Méthodes d'estimations le plus utilisées.

Remarques

- Ce cours utilise comme support le livre de Ian Sommerville *«Software Engineering»*, 9th edition.

Les logiciels et la société



Génie logiciel

Le **génie logiciel** est un domaine des sciences de l'ingénieur dont l'objet d'étude est la conception, la fabrication, et la maintenance des systèmes informatiques complexes. 3 févr. 2011

Génie Logiciel Avancé Cours 1 — Introduction - Stefano Zacchiroli
https://upsilon.cc/~zack/teaching/1011/gla/cours-01.pdf

Le génie logiciel touche au cycle de vie des logiciels. Toutes les phases de la création d'un logiciel informatique y sont enseignées : l'analyse du besoin, l'élaboration des spécifications, la **conceptualisation du mécanisme interne au logiciel** ainsi que les techniques de programmation, le développement, la phase de test et finalement la maintenance.
https://fr.wikipedia.org/wiki/Génie_logiciel

Mesure et Qualité des logiciels: Mesurer le logiciel !

Mesurer le logiciel !

- 80% des **coûts** du logiciel sont des coûts de maintenance
- Il y a en moyenne 55 **bogues** par 1000 lignes de code
- La méthode X permet d'accroître la **productivité** des études de 25%
- En consacrant plus de temps à la spécification, on diminue le **coût** total du projet
- Ce logiciel est **convivial, fiable, maintenable**
- Tel algo de tri un a une **efficacité** de *n* log *n*

Ces phrases expriment une notion de mesure!!!

L'acte de mesurer consiste à : *Affecter des valeurs (nombres, symboles) à des attributs d'entités du monde réel de manière à ce qu'on les décrive selon des règles clairement établies*

Mesurer le logiciel!

- Les exemples proposés ne correspondent pas complètement à cette définition :
  - On ne sait pas quelles sont les entités dont on parle
  - On ne sait pas quelle est la définition des attributs
  - On n'a pas exprimé les attributs en des termes mesurables
  - On ne sait pas ce que l'on désire mesurer

Pourquoi mesurer?

*La production du logiciel n'est pas contrôlée parce que l'on ne fait pas de mesure et que l'**on ne peut contrôler ce que l'on ne peut pas mesurer***

Tom DeMarco, 1982

Pourquoi mesurer ?

- Au niveau projet :
  - Déterminer une enveloppe budgétaire
  - Estimer le poids du projet en termes d'effort
  - Faire une estimation de la rentabilité de l'investissement
  - Evaluer une durée vraisemblable du projet
- Au niveau étape :
  - Ajuster le découpage
  - Sous-traiter
  - Prévoir les délais
  - Prévoir les ressources
- Au niveau phase :
  - Faire une planification précise
  - Annoncer un calendrier de remise des différents résultats intermédiaires
  - Prévoir et effectuer un suivi du projet ou sous-projet, pour surveiller les écarts
  - Prévoir l'affectation des ressources

Pourquoi mesurer? Besoin ?

- Le **coût** de différents processus pendant le développement (déterminer le prix)
- Mesurer la **productivité** du personnel (définir les salaires)
- Mesurer la **qualité** du logiciel (comparer les projets, faire des prévisions, se donner des objectifs d'amélioration)
- Définir des objectifs mesurables (ex: couverture de test à effectuer)
- Mesurer plusieurs des attributs et ressources d'un processus particulier (déterminer quels facteurs affectent le **coût** et la **productivité**)
- Evaluer l'**efficacité** de diverses méthodes et outils

Pourquoi mesurer? Besoin ?

- Mesurer les modifications faites pendant la conception, les **erreurs** trouvées en phase révision ou de test (surveiller la **qualité** des systèmes en évolution)
- Définir des attributs mesurables (spécifier des exigences de **qualité** et de performance qui soient testables)
- Mesurer des attributs de produits et de processus (obtenir une certification)
- Mesurer des attributs de produits existants et de processus en cours (faire des prévisions)

Evaluer
Prédire

Que mesure-t-on ?

- Coût
- bogues
- fiabilité
- maintenabilité
- convivialité
- efficacité
- ...

Les composants du coût d'un logiciel

- Coût du matériel
- Coût des logiciels de base
- Coût de la formation et des voyages
- Coût de l'effort
- Salaires des ingénieurs impliqués dans le projet
- Coût des bâtiments, climatisation, éclairage
- Coût des communications et des réseaux
- Coût des facilités partagées
- Coût des pensions, assurances maladies,…

Facteurs affectant le prix

- Opportunité du marché
- Estimation incertaine du coût
- Termes du contrat
- Changement rapide des besoins
- Situation financière du fournisseur

Productivité des programmeurs

- **Définition**: Mesure de la vitesse à laquelle les ingénieurs produisent des programmes et la documentation associée
- **Objectif**: mesurer le nombre de fonctionnalités utiles produites par unité de temps

Métriques de la productivité

- **Métriques associées à la taille du produit:**
  - Nombre de lignes de code
  - Nombre d'instructions
  - …
- **Métriques associées à la fonctionnalité du logiciel:**
  - Points de fonctions

Facteurs affectant la productivité

- Expérience dans le domaine
- Qualité du processus
- Taille du projet
- Assistance technique
- Environnement de travail

Qualité et productivité

- Les métriques basées sur le volume par unité de temps (Nombre de lignes de code, par exemple) sont imparfaites puisqu'elles ne prennent pas en compte la **qualité**
- En général, la **productivité** peut être améliorée au détriment de la **qualité**
- Les liens entre les métriques de **productivité** et les métriques de **qualité** ne sont pas faciles à déterminer

Mesure et Qualité des logiciels: Qualité du logiciel et métriques

Qualité: *Ce qu'est la qualité (du logiciel)*

En Génie Logiciel

La qualité d'un logiciel est son aptitude à satisfaire les besoins (exprimés ou potentiels) des utilisateurs
[Martin, 1987](NF X50-120)

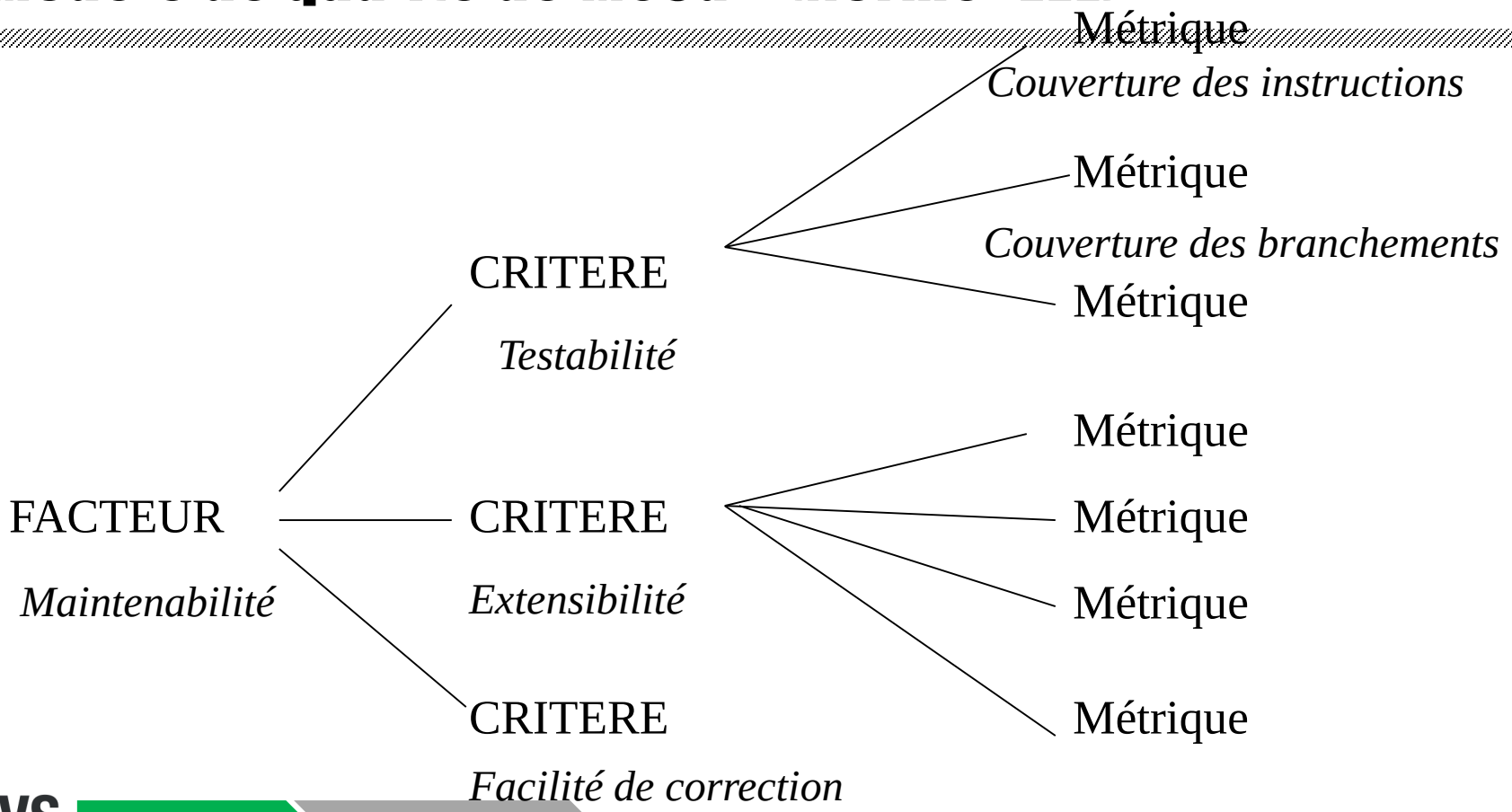
Qualité (Boehm)

- Un logiciel doit être :
  - Utilisable en l'état
  - Maintenable
  - Portable

Modèle de qualité

- **Facteurs de qualité** : Attributs **externes** du logiciel
  - fiabilité, facilité d'utilisation, maintenabilité,...
- **Critères de qualité** : Attributs **internes** du logiciel
  - la testabilité est un des critères de maintenabilité
- **Métriques** : attributs mesurables associés à chaque critère
  - Pour la testabilité : Degré de test mesuré par la couverture des instructions, la couverture des branchements, etc.

Modèle de qualité de McCall (Norme IEEE)



Définition de la mesure

- **Mesurer c'est affecter une valeur à un attribut d'une entité de telle manière que cela représente bien l'entité. Exemple** : On affecte la valeur 10 à l'attribut « temps passé en homme-mois » de l'entité « codage du programme P5 »
- 3 classes d'entités :
  - **Processus** : Activités que l'on peut bien identifier dans le développement du logiciel
    - Spécification, conception, codage
  - **Produits** : Résultats des processus
    - Dossier de spécifications fonctionnelles, code source d'un programme, plan de test
- **Projet/Ressources** : Entrées du processus
  - Personnel, matériel, outils (logiciels et matériels), méthodes

Définition de la mesure: Attributs internes et externes

- **Attributs internes** : Ceux qui peuvent être mesurés en termes de produits, processus, ressource
  - La ***longueur*** pour représenter la taille d'un produit : spécification, conception, code
  - La ***modularité***, le ***couplage,*** la ***cohésion*** d'une conception
  - La ***structuration du flot de contrôle*** du programme
  - La ***complexité cyclomatique*** du code
  - L'***effort*** en nombre de personnes et en temps pour réaliser tel processus
  - Le ***nombre d'incidents*** en cours de processus
- **Attributs externes** : Ceux qui peuvent seulement être mesurés relativement à la façon dont le produit, le processus, la ressource sont reliés à leur environnement
  - La ***maintenabilité*** d'une spécification, d'une conception, du code
  - La ***qualité*** de conception

Exemples de métriques

- **Lignes de code** : mesure de la taille d'un logiciel
- **Points fonctions** : mesure des fonctionnalités du logiciel
- **Homme-mois** : mesure de l'effort de développement

Effort ? Durée ? Taille ?...

- **Effort ou charge** : quantité de travail nécessaire, indépendamment du nombre de personnes qui vont réaliser ce travail
  - Permet d'obtenir un coût prévisionnel
  - S'exprime en homme/jour, homme/mois ou homme/année
  - Un homme/mois (HM) représente l'équivalent du travail d'une personne pendant un mois, généralement 20 jours
  - Un homme/mois (HM)=152 heures de travail par mois
  - **Exemple**:
    - Un projet de 60 mois/homme représente l'équivalent du travail d'une personne pendant 60 mois,
    - Si on évalue le coût du mois/homme à 50 KCFA en moyenne, le projet sera estimé à 3 MCFA

Effort ? Durée ? Taille ?...

On mesure la **taille** des projets à leur charge

- Si Charge < 6 HM : très petit projet
- Si 6 HM< Charge <12 HM : petit projet
- Si 12 HM< Charge <30 HM : projet moyen
- Si 30 HM < Charge < 100 HM : grand projet
- Si Charge > 100 HM : très grand projet

La **Durée** dépend du nombre de personnes

- 60 HM peut correspondre à :
  - 1 personne pendant 5 ans ,
  - 5 personnes pendant un an ,
  - 10 personnes pendant 6 mois
  - 60 personnes pendant 1 mois

L'Homme-Mois

- Unité de mesure de l'effort
  - Un homme pendant un mois
  - Deux hommes-mois = 2 hommes pendant 1 mois, ou 1 homme pendant deux mois

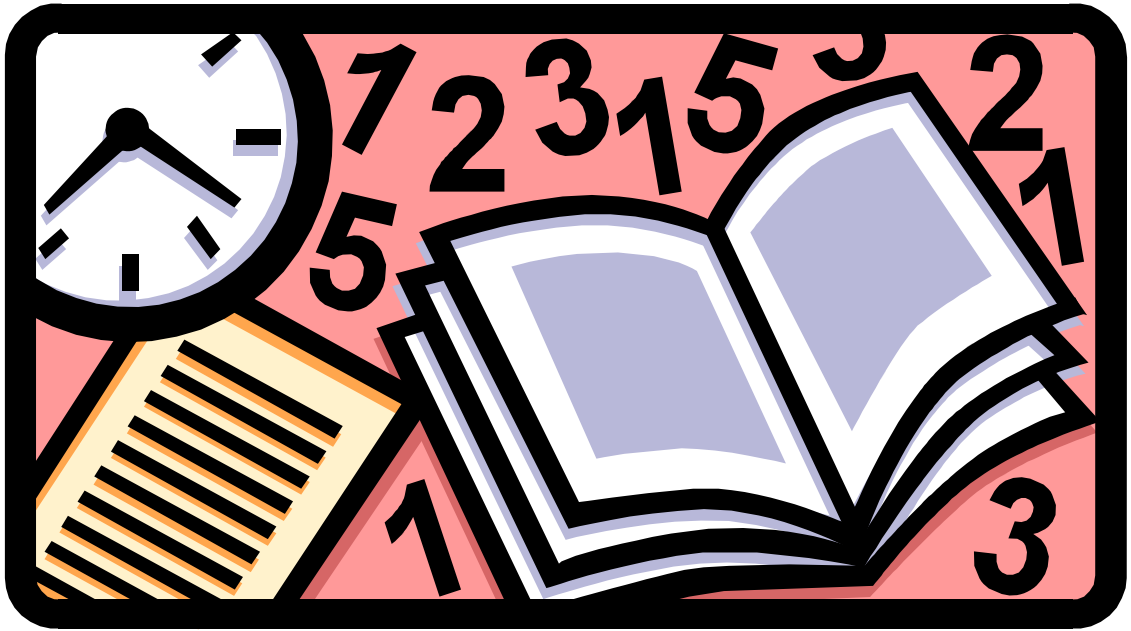
Selon Brooks : « *L'homme-mois comme unité pour mesurer la taille d'un travail est un mythe dangereux et trompeur. Il implique que les hommes et les mois sont interchangeables. Les hommes et les mois sont des biens interchangeables seulement lorsqu'une tâche peut être partitionnée entre plusieurs employés sans qu'il faille une communication entre eux* ».

Mesure et Qualité des logiciels:
Métrique logicielle - Les ligne de code

Les lignes de code

- Définition [Fenton 1991]
  - ***Une ligne de code est toute ligne du texte d'un programme qui n'est pas une ligne de commentaires ou une ligne blanche, sans considération du nombre d'instructions ou de fragments d'instructions dans la ligne. Sont incluses toutes les lignes contenant des en-têtes de programmes, des déclarations et des instructions exécutables et non exécutables.***

Exemple (Fenton 97, p. 250)



Problèmes avec les lignes de code

- Selon les définitions, le nombre de lignes d'un même programme peut varier de 1 à 5
- Que mesurer avec ce nombre ?
  - L'effort ?
    - Certaines lignes étant plus difficiles à coder que d'autres, il faudrait une pondération de ces lignes
  - Fonctionnalité du produit ?
  - Selon le langage, la même fonctionnalité prendra plus ou moins de lignes

Problèmes avec les lignes de code

- Qu'est-ce qu'une ligne de code?
- Quels programmes doivent être considérés comme faisant partie du système?
- L'hypothèse selon laquelle il existe une relation linéaire entre le volume de la documentation et la taille du système
- Plus le langage est de bas niveau, plus le programmeur utilisant ce langage est productif
- Plus le style est verbeux, plus la productivité est élevée

Exemple: Productivité comparée

Langage	Taille en lignes de code	Effort (en semaines)	Productivité
Assembleur	5000	28	714 lignes/mois
Langage de haut niveau	1500	20	300 lignes/mois

Pourquoi utiliser les lignes de code ?

- Des études statistiques ont montré que le nombre de lignes de code corrèle mieux que tout autre mesure (par exemple, mesure de HALSTEAD, de McCABE) avec les différentes autres données de mesures telles que celles sur l'effort de développement et de maintenance et sur les erreurs

Mesure et Qualité des logiciels: Métrique logicielle - Les points de fonctions (FP)

Les points de fonctions (FP)

- Le comptage d'objets de chacune des catégories est effectué
- la métrique des points de fonctions est obtenue en faisant la somme de chacun des nombres d'objets (de chaque catégorie) multiplié par la pondération associée à la catégorie correspondante
- Possibilité de prendre en compte la complexité du logiciel dans le calcul du nombre de points de fonctions (FP)
- BUT : fournir une mesure de la taille du produit qui soit disponible assez tôt dans le processus de développement et qui soit indépendant de la technologie
- Evite le problème de l'estimation de la taille en termes de lignes de code comme dans COCOMO

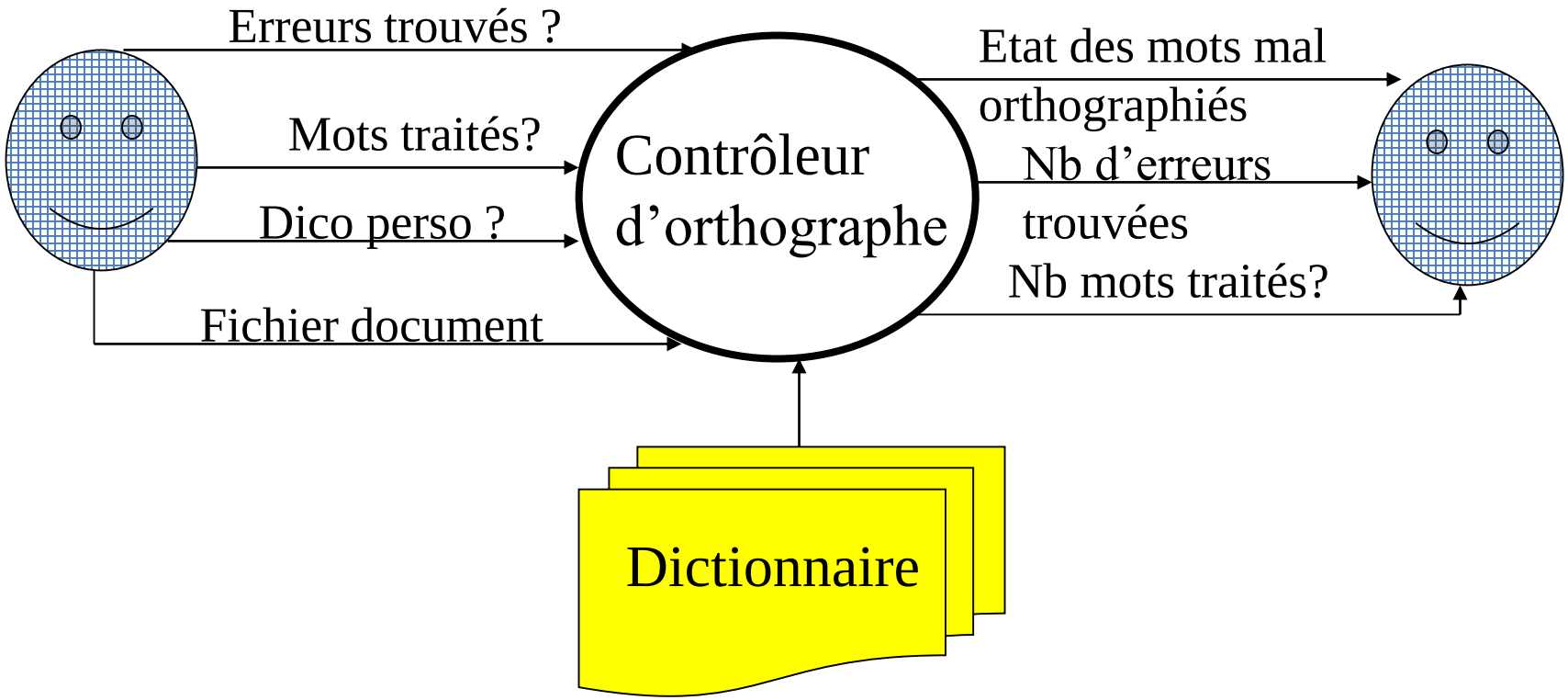
Les points de fonctions (FP)

- **Définition**: Mesure basée sur une combinaison des caractéristiques du logiciel
  - Les entrées/sorties externes
  - Les interactions utilisateurs (interrogations)
  - les interfaces externes
  - les fichiers utilisés par le logiciel
- Une pondération est associée à chacun de ces éléments

Les points de fonctions (FP)

- Le nombre de points de fonctions (FP) peut être utilisé pour estimer le nombre de lignes de code (LOC) en se basant sur le nombre moyen de lignes de code par Point de Fonction (FP) associé à un langage donné
- La métrique des points de fonction est très subjective et dépend des pondérations qui ne peuvent pas être calculées automatiquement

Exemple (Fenton, p.261)



Méthode des points fonctions : Principes

- Mesurer le "montant" de fonctionnalité dans un système décrit par une spécification.
- Faire une estimation à partir de la description externe du futur système, de ses "fonctions" ou composants "fonctionnels".
- On identifie cinq type d'unité d'œuvre et trois degrés de complexité (faible, moyen, élevée). A chaque type et chaque degré est affecté un nombre de points. La méthode permet de calculer, pour un projet donné, son poids en points de fonction.

Les composants fonctionnels

- *Entrées externes (external inputs) :* Eléments (items) fournis par l'utilisateur qui décrivent les différentes données orientées-application (file names, menu selection)
- *Sorties externes (external output) :* Eléments (items) fournis par l'utilisateur générant des données distinctes orientée-application (reports and messages).
- *Questions (inquiries) :* Entrants interactifs réclamant une réponse
- *Fichiers externes (external files) :* Machine-readable interfaces to other systems
- *Fichiers internes (internal files) :* Logical master files in the system

Table des facteurs de pondération

Type élément	Simple	Moyen	Difficile
Entrées externes	3	4	6
Sorties externes	4	5	7
Questions	3	4	6
Fichiers externes	7	10	15
Fichiers internes	5	7	10

Etape 1 : *Compter le nombre d'éléments pour chaque type de composant*

Type	Nombre	Description
Entrées externes		
Sorties externes		
Questions		
Fichiers externes		
Fichiers internes		

Etape 2 - 3:

- Affectation d'un niveau de complexité à chaque élément pris dans l'ensemble {simple, moyen, élevé}
  Comptage du nombre d'éléments de chaque type par valeur de complexité

Type	Nombre	Description
Entrées externes		
Sorties externes		
Questions		
Fichiers externes		
Fichiers internes		

Etape 4 : Calcul UFC

- Multiplier chaque valeur de la table obtenue (étape 3) par le "facteur de pondération" correspondant pris dans la table des facteurs de pondération .
- Faire la somme des 15 produits. On obtient le nombre de points de fonction non ajusté : (UFC)

Etape 5 : Facteur de complexité technique (TCF)

Albrecht prend en compte 14 facteurs contribuant à la complexité

- F1 : Fiabilité des sauvegardes et reprises
- F2 : Communications de données
- F3: Fonctions distribuées
- F4 : Performance
- F5 : Forte charge de la configuration
- F6 : Entrées de données en ligne
- F7 : Facilité d'utilisation
- F8 : Mise à jour en ligne
- F9 : Interface complexe
- F10 : Traitement complexe
- F11 : Réutilisabilité
- F12 : Facilité d'installation
- F13 : Sites multiples
- F14 : Facilité de modification

Etape 5

- Chaque facteur est évalué sur une échelle ordinale de 0 à 5 où 0 signifie "sans rapport" et 5 signifie " essentiel".
- On supposera :

  F3 = F5 = F9 = F11 = F12 = F13 = 0

  F1 = F2 = F6 = F7 = F8 = F14 = 3

  F4 = F10 = 5

  → Calcul du TCF

Etape 6 : Calculer FP

- FP = UFC x TCF

- On en déduit l'Effort = FP x Nb j/h par FP
  - (Si l'on sait combien un point fonction prend de jours/homme)
  - (Habrias 94, p. 67) (Boehm 00, p. 470) (Fenton 97, p. 415)

UVS
UNIVERSITE VIRTUELLE DU SENEGAL

Mesure et Qualité des logiciels:
Estimation

L'estimation: *Les règles d'estimation de DeMarco*

- Estimer est différent de régurgiter
- Estimer est différent de négocier
- Les estimations ne sont pas sujettes à marchandage
- Estimer est différent de diviser une durée fixe en parties
- Un dérapage dans une phase du projet implique un dérapage proportionnel dans toutes les phases qui suivent
- Si vous désirez que quelqu'un vous fournisse une estimation significative, ne lui soufflez pas la réponse
- Une estimation de planning utile est une projection dont la probabilité n'est pas plus pessimiste qu'optimiste
- Le ration entre l'estimation de planning la plus optimiste et l'estimation utile est moyennement uniforme pour tout individu
- Estimez avec un comité

Techniques d'estimation

- Affectation de prix pour gagner (La meilleure offre)
- Loi de Parkinson
- Estimation par analogie
- Jugement d'experts
- Méthode de Répartition proportionnelle : Estimation descendante/ascendante /dynamique
- Estimation globale/détaillée
- Modélisation algorithmique

Affectation des prix pour gagner: *La meilleure offre ou méthode du marché*

- **Principe**: le projet coûte ce que le client est prêt à payer
- **Avantages**: Le contrat sera gagné
- **Inconvénients**: La probabilité que le client obtienne le logiciel qu'il désire est très faible. Les coûts ne reflètent pas le travail requis.

Loi de Parkinson

- **Principe**: le projet utilise toutes les ressources disponibles
  - Loi de Parkinson : « le travail se dilate jusqu'à remplir le temps disponible »
- **Avantages**: pas d'excès dans les dépenses
- **Inconvénients**: projets souvent non achevés

En absence d'estimation, la vitesse d'avancement est ajustée à la durée de la dispo des individus (de la ressource)
Souvent les entreprises préfèrent ne pas faire d'estimation de leurs projets SI considérant que « cela ne tombe jamais juste ».
Mais la « loi de Parkinson » indique que sans éval , on consomme en général bcp plus de temps et que le travail s'achève difficilement

Estimation par analogie

- **Principe**: le coût du projet est calculé en le comparant à d'autres projets similaires dans le même domaine d'application
- **Avantages**: Estimation précises si des données concernant des projets similaires sont disponibles
- **Inconvénients**:
  - Impossible si des projets similaires n'étaient pas réalisés
  - Nécessite la maintenance systématique d'une base de données de coût de projets

Jugement d'experts

- **Principe**: un ou plusieurs experts à la fois dans le domaine d'application et en processus de développement utilisent leur expertise pour faire des prévisions de coût. Le processus est itéré jusqu'à ce qu'un consensus soit atteint
- **Avantages**:
  - méthode relativement bon marché
  - estimations précises si les experts ont de l'expérience dans des logiciels similaires
- **Inconvénients**: estimations non précises s'il y a assez peu d'expertise

Estimation ascendante

- **Principe**:
  - Commencer au niveau le plus bas du logiciel
  - Estimer le coût de chaque composant individuellement
  - Ces coûts sont additionnés pour obtenir une estimation du coût global du logiciel
- **Avantages**: Méthode précise si la conception a été effectuée avec suffisamment de détails
- **Inconvénients**: Possibilité de sous-estimer les coûts des activités au niveau global du logiciel: intégration, documentation,...

Estimation descendante

- **Principe**: Partir du niveau global du logiciel jusqu'au niveau fonctionnel
- **Avantages**: Prendre en compte des coûts globaux comme celui de l'intégration, de la gestion de configuration, de la documentation
- **Inconvénients**: Possibilité de sous-estimer le coût d'apporter des solutions à des problèmes techniques de bas niveau

Modélisation algorithmique

- L’effort de développement, le coût, le temps ... sont estimés comme une fonction mathématique des attributs du logiciel, du projet et du processus de développement.
- Les valeurs de ces attributs sont estimées par des gérants des projets
- La fonction est déterminée par une étude empirique de données concernant le logiciel
- L'attribut le plus communément utilisé dans l'estimation de l’effort de développement du logiciel est la taille du produit exprimée en nombre de lignes de code
- La plupart des modèles sont similaires mais basés sur différents types d'attributs: taille, temps de développement, ...

Structure des modèles algorithmiques d'estimation

- Modèles dérivés sur la base d'analyse économétrique de données collectées sur des projets logiciels passés
- Forme générale: $E = A + B\,(EV)^C$ avec:
  - E: effort en hommes-mois
  - EV: variable d'estimation (nombre de milliers de lignes de code ou KLOC, nombre de points de fonctions, nombre de fichiers, nombre de modules...)
  - A, B, C : constantes estimées empiriquement

Exemple de modèles algorithmiques d'estimation

- $E=5.2\ (KLOC)^{0.91}$ (Walston-Felix)
- $E=5.5+0.73\ (KLOC)^{1.16}$ (Bailey-Basili)
- $E=2.4\ (KLOC)^{1.05}$ (Boehm COCOMO simple)
- $E=5.288\ (KLOC)^{1.047}$ (Doty si KLOC >9)
- $E= -13.39+0.0545\ FP$ (Albrecht-Gaffney)
- $E=60.62x7.728x10^{-8}xFP^3$ (Kemerer)
- $E=585.7+15.12\ FP$ (Matson-Barnett-Mellichamp)

Modèles algorithmiques d'estimation

Autres métriques :

- Documentation en *nb* de pages = X $(KLOC)^{Y}$
- Durée du projet en mois = $\alpha(KLOC)^{\beta}$
- Durée du projet en mois = Δ Homme-Mois$^{\delta}$
- Temps de développement = T Hommes-Mois $^{\tau}$

Tests de modèles: *Expérience de MOHANTY*

- Projet « hypothétique »
  - environ 36 KLOC (instructions exécutables)
  - coût 50 000 dollars par an
- Estimations de coûts obtenues avec les différents modèles :
  - de 365 000 dollars à 2 766 667, soit **plus de 7,5 fois plus** à partir des mêmes données.

Tests de modèles: *Expérience de KEMERER*

- Comparaison de modèles (domaine des applications de gestion):
  - COCOMO et PUTNAM
  - 2 autres modèles qui ne se basent pas sur le nb de lignes de code en entrée

→ Coûts prédits : de 230 H/M à 3857 H/M pour le même projet

Tests de modèles: *Expérience de RENOUS (1988)*

- Ratios relevés pour un logiciel de gestion écrit en COBOL :
  - 213 lignes par jour en programmation (130000 lignes en 29 H/M : 31-2 gestion de projet)
  - 151 lignes par jour sur l'ensemble des deux phases conception et programmation et intégration
  - 101 lignes par jour sur l'ensemble conception, programmation, intégration

- *Résultat de la comparaison*

	COCOMO	ELYS	SEL	W et F	Projet Réel
Effort (H/M)	276	77	128	436	80
Délai(M)	15	26	16	14	18

Mesure et Qualité des logiciels: Le modèle COCOMO (Construtive COst MOdel)

Le modèle COCOMO *(Construtive COst MOdel)*

- Développé à la firme TRW (organisme du DoD, USA) par B.W. Boehm et son équipe
- **Référence:** Boehm B.W., « Software Engineering Economics », Prentice-Hall, Englewood-Cliffs, 1981

Le modèle COCOMO: *Les hypothèses*

- Un informaticien chevronné sait plus facilement donner une évaluation de la taille du logiciel à développer que faire une estimation du travail nécessaire
- Il faut toujours le même effort pour écrire un nombre donné de lignes de programme, quel que soit le L3G employé
- Bonne gestion du projet logiciel
- Stabilité des besoins issus de la phase de définition des besoins

Le modèle COCOMO

- Etude de la corrélation entre la taille du logiciel et la charge consommée
- Basé sur une base de données de plus de 60 projets différents
- Les estimations ne tiennent pas compte de certains efforts comme la formation des utilisateurs, l'installation du logiciel et la conversion

Le modèle COCOMO de base

- Bien adapté pour des estimations rapides du coût dès les premières phases du cycle de vie du logiciel
- Applicable aux projets logiciels de petite ou moyenne taille développés en interne au sein des organisations, l'aide d'environnements de développement « familier »
- N'inclut pas les facteurs liés aux contraintes matérielles, aux attributs du personnel, à l'utilisation d'outils et techniques modernes de génie logiciel

Le modèle COCOMO de base: *Trois modes*

- Organique (Simple)
- Semi-détaché (Moyen)
- Embarqué ou Intégré (Complexe)

Le mode organique : *Projet simple*

- Logiciel de moins de 50 000 instructions (*)
- Petites équipes de développement
- Applications maîtrisées
- Problèmes bien compris
- Pas de besoins non fonctionnels difficiles
- Spécifications stables

Le mode semi-détaché: *Projet moyen*

- Logiciel comportant entre 50 000 et 300 000 instructions (*)
- Expérience variée des membres de l'équipe de projet
- Possibilité d'avoir des contraintes non fonctionnelles importantes
- Type d'application non maîtrisée par l'organisation (nouvelles technologies par exemple)

Le mode embarqué (intégré): *Projet complexe*

- Logiciel comportant plus de 300 000 instructions (*)
- Contraintes serrées

- L'équipe de projet nombreuse a, en général, peu d'expérience de l'application

- Problèmes complexes

Les formules du modèle COCOMO: *Notations*

- HM : charge de développement exprimée en hommes-mois
- TDEV: temps de développement exprimé en mois
- KLOC: Millier de lignes de code
- N: nombre de personnes requises

Modèle COCOMO de base: Effort

- **E = A (Taille)^B** mesuré en HM
- **Taille** est mesuré en KLOC
- A et B sont des paramètres dépendant :
  - du modèle COCOMO (simple, intermédiaire, détaillé)
  - du type de logiciel (organique, semi-détaché, intégré)

KLOC = millier d'instructions sources livrées ou KDSI (Kilo Delivered Source Instructions) = nombre de lignes de code source, commentaires exclus

Mode organique	:	A=2.4	B=1.05
Mode semi-détaché	:	A=3	B=1.12
Mode embarqué	:	A=3.6	B=1.20

Modèle COCOMO de base: *Temps de développement*

Le temps de développement est fonction de l'effort et non du nombre de personnes travaillant sur le projet :

- **Mode organique**: $TDEV=2.5(E)^{0.38}$
- **Mode semi-détaché**: $TDEV=2.5(E)^{0.35}$
- **Mode embarqué**: $TDEV=2.5(E)^{0.32}$
- **Personnel requis**: $N=E/TDEV$

Le modèle COCOMO en résumé

- Charge en H/M = a (KDSI) ^{b}
- Délai normal en mois = c (charge en H/M) ^{d}

Type projet	Charge en H/M(Coût)	Délai normal(TDEV)
Simple / Applications < 50 KDSI	a = 2,4 b = 1,05	c = 2,5 d = 0,38
Moyen / Utilitaires < 300 KDSI	a = 3 b = 1,12	c = 2,5 d = 0,35
Complexe / Programmes systèmes > 300 KDSI	a = 3,6 b = 1,2	c = 2,5 d = 0,32

Exemple

Projet en mode organique (simple)

- Taille = 32 000 ISL
- Effort = 2,4 KISL ^{1,05} = 2,4 (32) ^{1,05} = 91 HM
- TDEV = 2,5 (Effort)^{0,38} = 2,5 (91) ^{0,38} = 14 mois
- Taille moyenne de l'équipe : 91/12 = 5,6 personnes (Si l'on suppose que l'on travaille 12 mois)

Remarques sur le modèle COCOMO de base

- Le temps de développement est une fonction de l'effort global de développement et non pas de la taille de l'équipe
- La prise en compte de la disponibilité du personnel par le modèle n'est pas claire

Le modèle COCOMO intermédiaire

- **Principe:** estimation intermédiaire modifiant l'estimation brute fournie par le modèle COCOMO de base en se servant des attributs (facteurs de coût):
  - du logiciel
  - du matériel
  - du projet
  - du personnel

Le modèle COCOMO intermédiaire

- **Effort = A Taille^{B} m(X)** mesuré en HM
- Où
  - Taille est mesuré en KLOC
  - A et B sont des paramètres dépendant :
    - Du modèle COCOMO (simple, intermédiaire, détaillé)
    - Du type de logiciel (organique, semi-détaché, intégré)
  - m(X) est le facteur d'ajustement de l'effort

Mode organique	:	A=3.2	B=1.05
Mode semi-détaché	:	A=3	B=1.12
Mode embarqué	:	A=2.8	B=1.20

Le modèle COCOMO intermédiaire: Facteur d'ajustement de l'effort

$$m(X) = \prod_{j=1}^{j=15} m(x_j)$$

- x_j est un facteur de coût (cost driver)
- $M(x_j)$ est le poids affecté au facteur de coût xj
  Ce poids va de 0,70 à 1,66
  $M(x_j)$= 1 pour tout x_j valorisé comme « nominal »

COCOMO intermédiaire (1981): Facteur d'ajustement de l'effort

- Exemple de poids affectés à des facteurs de coût

	Très basse	basse	nominale	haute	Très haute	Extra haute
Fiabilité	0,75	0,88	1	1,15	1,40	-
Complexité	0,70	0,85	1	1,15	1,30	1,65
Contr.d'exécution en temps	-	-	1	1,11	1,30	1,66
Contr.sur délai	1,23	1,08	1	1,04	1,10	

COCOMO (version 2000): *Facteur d'ajustement de l'effort*

- *Exemple de poids affectés à des facteurs de coût*

	Très basse	basse	nominale	haute	Très haute	Extra haute
Fiabilité	0.82	0.92	1.00	1.10	**1.26**	
Complexité	0.73	0.87	1.00	1.17	1.34	1.74
Contr.d'exécution en temps			1.00	1.11	1.29	1.63
Contr.sur délai	1.43	1.14	1.00	1.00	1.00	

Les attributs du produit

- Besoin en fiabilité : RELY
- Taille de la Base de Données : DATA
- Complexité du produit : CPLX
- RUSE (Voir dans Costar)
- DOCU (Voir dans Costar)

Les attributs du matériel

- Contraintes du temps d'exécution : TIME
- Contraintes de la mémoire : STOR
- Instabilité de la machine virtuelle : VIRT
- Temps de retournement (ou temps de restitution des travaux) : TURN (Computer Turnaround Time)

Les attributs du projet

- Pratiques de la programmation moderne (POO, Programmation Evénementielle,...) : MODP
- Utilisation d'outils de génie logiciel (CASE) : TOOL
- Contraintes de développement (délai, budget,...) : SCED

Les attributs du personnel

- Compétence des analystes : ACAP
- Compétence des programmeurs : PCAP
- Expérience dans l'utilisation du langage de programmation : LTEX
- Expérience dans le domaine de l'application : APEX
- Expérience dans l'utilisation du matériel : PLEX

Le modèle COCOMO détaillé

- Description du logiciel comme une hiérarchie de modules, sous-systèmes et systèmes
- Estimation de l'effort de développement par phase: Conception globale, Conception détaillé, Codage, Intégration, Tests d'acceptation
- A chaque attribut décrit ci-dessus est associé un facteur de coût propre à chaque phase

Démarche

- Estimation de la taille (Nb d'instructions sources)
- Calcul de la charge « brute »
- Sélection des facteurs correcteurs
- Application des facteurs correcteurs à la charge brute pour obtenir la charge « nette »
- Evaluation du délai normal

COCOMO et la maintenance

Effort annuel de maintenance (en HM)
=
Taux annuel de modification (% de code modifié)
x
Temps de développement (en HM)

NB: On peut tenir compte des facteurs de coût qui ont un impact sur le coût de la maintenance et calculer un facteur d'ajustement d'effort

Réglage du modèle COCOMO

- Modification des paramètres du modèle de COCOMO en prenant en compte des spécificités, des contraintes et des priorités de l'organisation
- Nécessité d'une base de données significative contenant des informations détaillées sur le coût du logiciels au sein de l'organisation

Facteurs non pris en compte par le modèle COCOMO

- Volatilité des besoins
- Disponibilité du personnel
- Qualité du management
- Qualité de l'interface homme-machine
- ...

Critiques de COCOMO

- Fenton remarque que la définition de la taille dans COCOMO inclut la notion d'effort
- Des études sur les écarts entre l'estimation du nombre de lignes de code par des gestionnaires et la valeur réelle ont montré que la déviation moyenne était de 64% de la taille réelle et que seulement 25% des estimations étaient à 25% de la taille réelle.
- Une étude sur les outils logiciels d'estimation ont montré qu'ils ne font pas mieux, la déviation pouvant aller de 100 à 200%

Evolutions de COCOMO

- COCOMO II
  - http://sunset.usc.edu/COCOMOII
  - « Software Cost Estimation with COCOMO II », Boehm et al. , Prentice Hall, 2000.
- Propose un affinement des paramètres de contexte et inclut des projets au stade de l'analyse, en s'appuyant sur la méthode des points fonctions

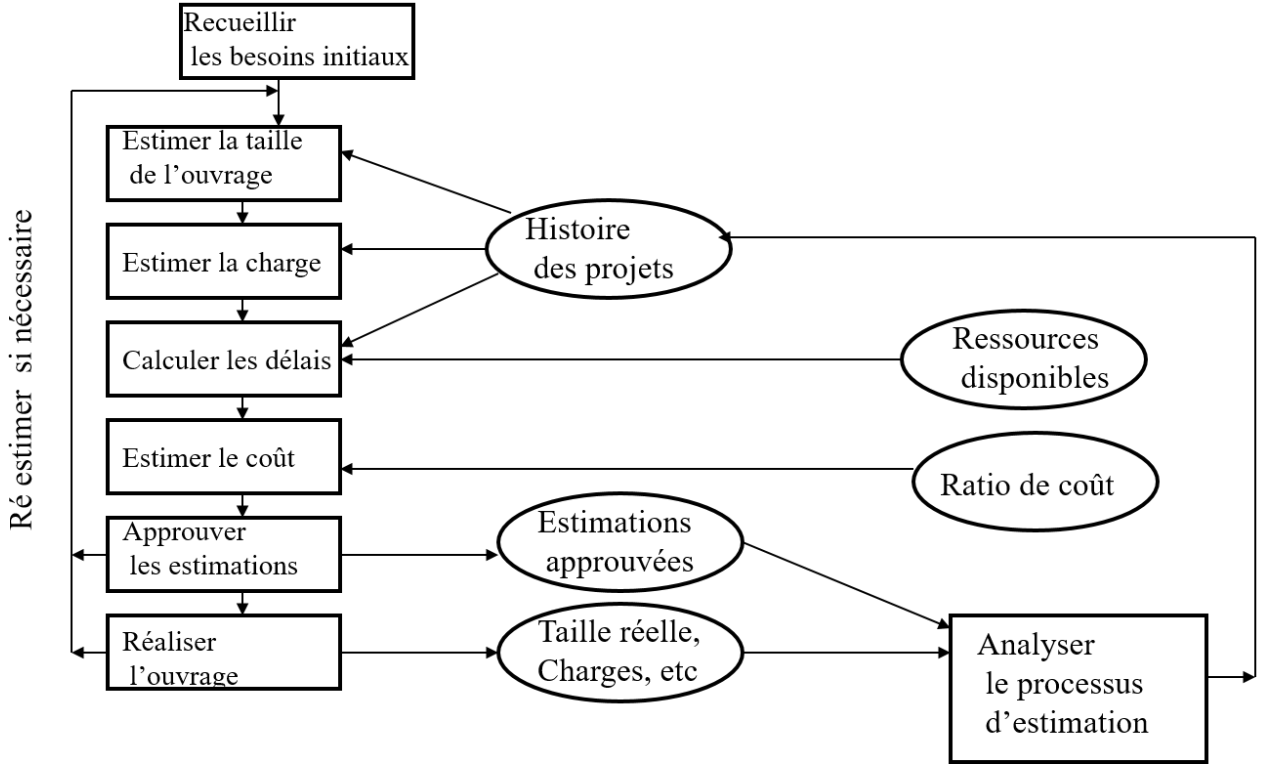
UVS
UNIVERSITE VIRTUELLE DU SENEGAL

Mesure et Qualité des logiciels:
Finalement...

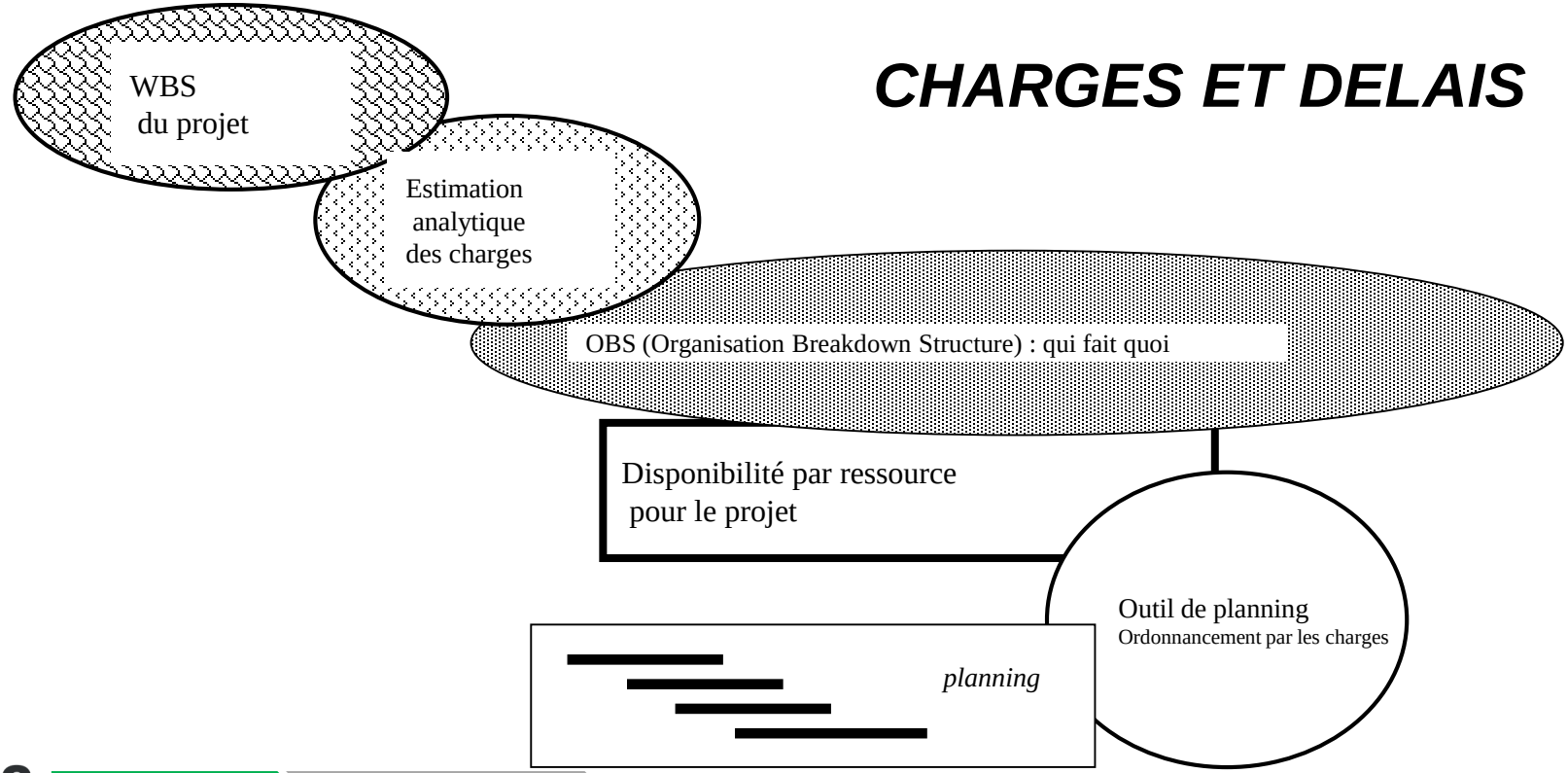
Quelle méthode d'estimation choisir?

- Chaque méthode a ses avantages et ses inconvénients
- L'estimation du coût du logiciel doit se baser sur plusieurs méthodes
- Le fait que les résultats fournis par les différentes méthodes sont sensiblement différents montre qu'il y a un manque d'information. Dans ce cas, des actions doivent être entreprises pour obtenir plus d'information permettant d'améliorer la précision des estimations
- L'affectation du coût pour gagner est parfois la seule méthode possible

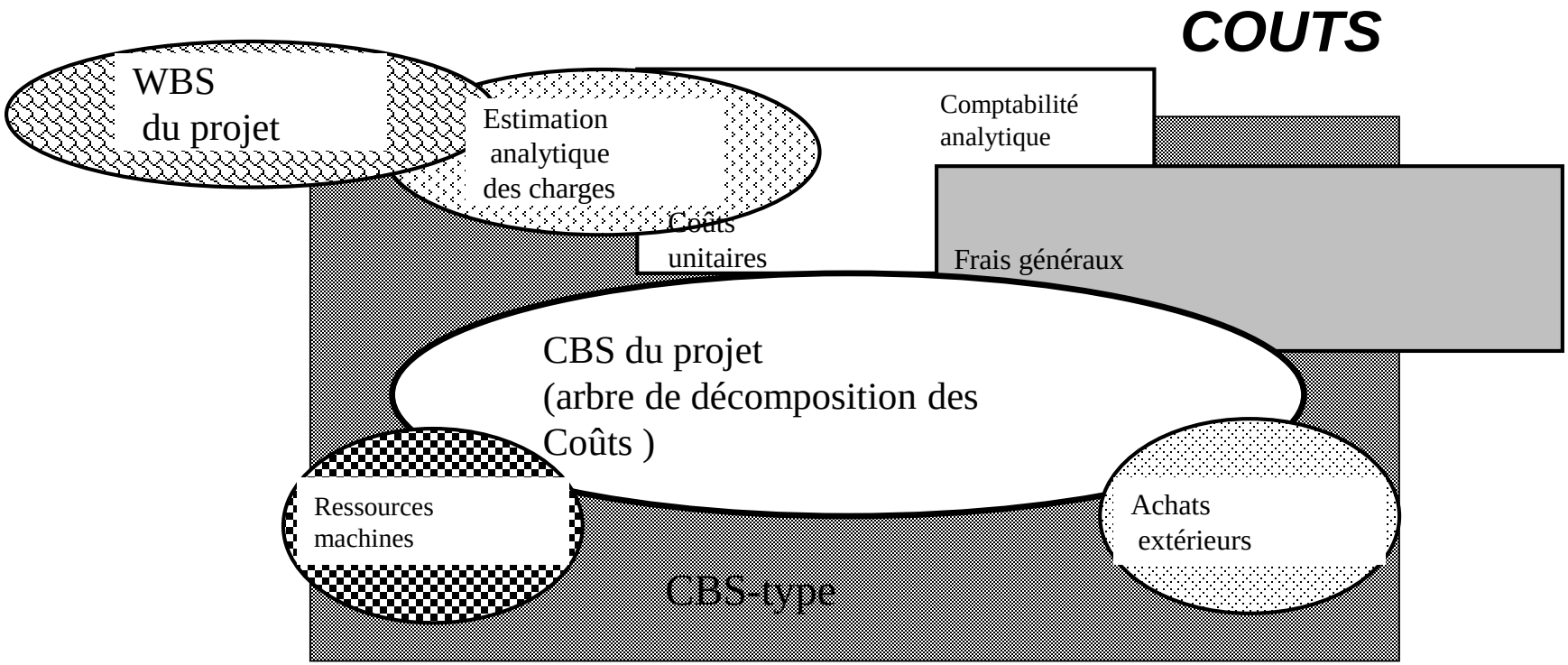
Processus d'estimation d'un projet



Estimation opérationnelle analogique
PLAN DE DEVELOPPEMENT



Estimation opérationnelle analogique: PLAN DE DEVELOPPEMENT



Conclusion

- Estimer le coût du projet pour le fournisseur puis décider du prix facturé au client
- L'estimation algorithmique du coût est difficile puisqu'elle repose sur des estimations des attributs du logiciel final
- Importance des modèles d'estimation du coût du logiciel pour le management: moyen de comparaison entre plusieurs options de développement
- Le temps requis pour développer un projet n'est pas simplement proportionnel à la taille de l'équipe de projet
- Chaque organisation doit déterminer ses propres attributs et ses propres multiplicateurs en fonction de ses spécificités, ses priorités et ses contraintes

Conception et Architectures Logicielles:
A Retenir...

A retenir…

- Une architecture logicielle est une description de la manière dont un système logiciel est organisé. Les propriétés d'un système, telles que les performances, la sécurité et la disponibilité, sont influencées par l'architecture utilisée.
- Les décisions de conception architecturale comprennent les décisions relatives au type d'application, à la distribution du système, aux styles architecturaux à utiliser et aux moyens de documenter et d'évaluer l'architecture.


thank you
danke
謝謝
рахмат
ngiyabonga
teşekkür ederim
gracias
merci
obrigado
dziękuję
bedankt
mauruuru
hvala
спасибо
Баярлалаа
faafetai lava
kiitos
dankie
dhanyavad
nanni
nandri
köszönöm
bayarlalaa
grâçie
enkosi
vinaka
спасибі
blagodaram
mersi
kia ora
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
хвала
asante
manana
obrigada
murakoze
tenki
chokrane
djiere dieuf
tau
mochchakkeram
дякую
mamnun
akun
dankon
aciù
chnorakaloutioun
gralias ago
gràcies
sulpáy
go raibh maith agat
sukriya
kop khun krap
taiku
takk
dakujem
trugarez
arigatô
ありがとう
grazie
tanemirt
rahmet
najis tuke
terima kasih
dhanyavadagalu
shukriya
мерси
merce
diolch
xièxie
감사합니다
ευχαριστώ
তোমাকে ধন্যবাদ
kam sah hamnida
didi madloba
sagolun
sobodi
dekuji
mèsi
rahmat