Techniques de gestion de projets

Chouki Tibermacine
Chouki.Tibermacine@umontpellier.fr

POLYTECH MONTPELLIER

Plan du cours

1. Notions générales

2. Familles de méthodes d'estimation

Plan du cours

1. Notions générales

2. Familles de méthodes d'estimation

Notion de charge

- Ne pas la confondre avec la notion de durée
- Charge = quantité de travail nécessaire à la réalisation du projet, indépendemment du nombre de personnes qui vont réaliser le travail
- Elle permet d'estimer le coût prévisionnel du projet
- Elle est exprimée en jour-personne, mois-personne ou année-personne
- Un mois-personne = travail d'une personne pendant un mois (20 jours environ)
- Projet de 60 mois-personne = travail d'une personne pendant 60 mois $\Rightarrow$ coût du projet = 900K€ (coût mois-personne=15k€)

Charge d'un projet ≡ sa taille

Ordres de grandeur

- Très petit projet : charge < 6 mois-personne
- Petit projet : charge entre 6 et 12 mois-personne
- Projet moyen : charge entre 12 et 30 mois-personne
- Grand projet : charge entre 30 et 100 mois-personne
- Très grand projet : charge > 100 mois-personne

Notion de durée

- La durée dépend du nombre de personnes (approx. égale à la charge divisée par le nombre de personnes)
- Ex : charge = 60 mois-personnes
  - 1 personne pendant 5 ans
  - cinq personnes pendant 1 an
  - 10 personnes pendant 6 mois
  - 60 personnes pendant 1 mois

Pourquoi et à quel niveau estimer les charges ?

- Pour tout le projet (mois-personne) :
  - déterminer une enveloppe budgétaire
  - voir ce que “pèse” le projet en termes d’effort
  - faire une estimation de la rentabilité de l’investissement
  - évalue une durée vraisemblable du projet
- Pour une phase ou une activité dans le projet (mois/semaine-personne) :
  - ajuster le découpage et sous-traiter
  - prévoir des délais et des ressources (surveiller les écarts)
  - annoncer un calendrier de remise
- Pour une tâche (jour-personne) : évaluer chaque tâche

Importance des méthodes d'estimation

- Constat : estimation des charges tombe rarement juste
- Mais sans cette évaluation, on consomme en général beaucoup plus de temps (Loi de Parkinson)
- Loi de Parkinson (1955) : "Le travail se dilate jusqu'à remplir le temps disponible"
- Méthode du marché : la charge correspond au prix à proposer pour remporter l'appel d'offre (travailler à perte, en espérant remporter d'autres contrats plus tard)

  - Cas célèbre d'un projet US Air Force : estimation à 1500k$
    - remporté par une société de services pour 400k$
    - coût total à la fin : 3700k$

Plan du cours

1. Notions générales

2. Familles de méthodes d'estimation

Cinq familles de méthodes d'estimation

Des méthodes pas forcément concurrentes (qui peuvent se compléter dans le cycle de vie d'un projet)

1. Jugement d'expert
2. Estimation par analogie
3. Estimation ascendante
4. Estimation paramétrique
5. Estimation probabiliste

1. Jugement d'expert

- Un expert = un consultant ou chef de projet expérimenté
- Méthode utilisée lorsqu'on a peu d'éléments pour l'estimation
- Exemple : méthode Delphi (1948) ⟶ démarche pour mettre en commun des jugements d'experts
- Méthode Delphi (confrontation collective des estimations) :
  1. chaque expert propose une estimation
  2. les jugements sont rendus publics mais restent anonymes (possibilité pour chaque expert de confirmer ou modifier, en fonction des autres jugements)
  3. les nouvelles estimations sont dévoilés (chacun peut justifier)
  4. chacun des experts propose une révision pour converger

2. Estimation par analogie

- S'appuyer sur des projets similaires (internes ou externes)
- Situer le projet à estimer en fonction de ses caractéristiques aux projets les plus proches (faire une moyenne des charges, par ex.)
- Exemples :
  1. Méthode de répartition proportionnelle
  2. Méthode des ratios

2.1. Méthode de répartition proportionnelle

- Charge de chaque phase d'un projet correspond à un pourcentage de la charge globale, avec un intervalle de tolérance
- Méthode utilisée de trois façons :
  1. On a fait une estimation de la charge globale, que l'on cherche à répartir dans le temps
  2. On a évalué l'une des phases, en utilisant une autre méthode, et on veut déduire la charge des autres phases
  3. On est en cours de déroulement du projet, on a observé le temps déjà consommé et on veut estimer celui des phases à venir

2.1. Méthode de répartition proportionnelle -suite-

- Il existe des pourcentages indicatifs pour des cycles de vie de référence (c'est une base qu'il faudra adapter à chaque projet)

Pour le modèle RUP [1] :

1. Opportunité : 5% charge et 10% durée
2. Élaboration : 20% charge et 30% durée
3. Construction : 65% charge et 50% durée
4. Transition : 10% charge et 10% durée

1. David West. Planning a Project with the Rational Unified Process. Rational Software White Paper, 2002.

2.1. Méthode de répartition proportionnelle -suite-

Pour un cycle de vie classique :

1. Étude préalable : 10% du total du projet (hors mise en oeuvre)
2. Étude détaillée : 20 à 30% du total du projet
3. Étude technique : 5 à 15% de la charge de réalisation
4. Réalisation : deux fois la charge d'étude détaillée

2.2. Méthode des ratios

- Variante de la méthode précédente qui vise la relation entre deux activités
- Utilisée souvent dans l'estimation des charges complémentaires au développement (recette, documentation, ...)
- Répartition proportionnelle des charges complémentaires de certaines activités :
  - L'activité "Recette" a un ratio de 20% de la charge de réalisation
  - L'activité "Documentation utilisateur" a un ratio de 5% de la charge de réalisation

3. Estimation ascendante (évaluation analytique)

- Méthode souvent utilisée pour estimer la charge de réalisation (une fois qu'on a une visibilité sur les modules à développer)
- Estimation des charges des tâches, activités, ...et ensuite faire les totaux pour le projet complet
- Méthode s'appuie sur une typologie des modules (nature, degré de difficulté, ...)
- Exemples de "poids" de certains modules logiciels (en jour-personne) :
  - Menu : 0.25 (Facile), 0.5 (Moyen), 1 (Difficile)
  - Consultation : 1 (Facile), 2.5 (Moyen), 4 (Difficile)

3. Estimation ascendante (évaluation analytique)

- On multiplie le nombre de modules à réaliser par les poids correspondants
- A cela, on ajoute les charges : i) des tests d'intégration (10% de la charge de réalisation) et ii) d'encadrement (20% de la charge de réalisation)

4. Estimation paramétrique

- Déterminer des **unités d'oeuvre** et leur affecter à chacune un coût standard (en divisant un coût global précédemment mesuré par le nombre total d'unités)
- Exemples : Modèle Cocomo et la méthode des points de fonction
- Schéma général de ces méthodes :
  1. dénombrement des unités d'oeuvre (output : taille du projet)
  2. application des poids standards (output : charge brute)
  3. application des facteurs correcteurs (output : charge ajustée)

4. Estimation paramétrique : Modèle Cocomo

- Cocomo (*COnstructive COst MOdel*) : proposé en 1981 par B. W. Boehm (même personne qui a développé le modèle en spirale)
- Le modèle repose sur deux hypothèses :
  1. un informaticien chevronné sait plus facilement évaluer la taille du logiciel à développer qu'estimer le travail nécessaire
  2. il faut toujours le même effort pour écrire un nombre donné de lignes de programme
- L'auteur de la méthode a calculé des coefficients de corrélation entre taille du logiciel et charge consommée pour une centaine de projets déjà réalisés
- Paramètre utilisé dans ce modèle = instruction dans le code source

Modèle Cocomo -suite-

- Charge en mois-personne = A (SLOC)^B
  où SLOC : *Source Lines of Code*
- Durée normale en mois = c (Charge en mois-personne)^d
- Formules selon le type de projet :

Type de projet	**Charge en mois-personne**	**Durée en mois**
Simple	charge = 2.4 (SLOC)$^{1.05}$	durée = 2.5 (charge) $^{0.38}$
Moyen	charge = 3 (SLOC)$^{1.12}$	durée = 2.5 (charge) $^{0.35}$
Complexe	charge = 3.6 (SLOC)$^{1.2}$	durée = 2.5 (charge) $^{0.32}$

- Exemple : un projet estimé à 40 000 instructions à développer
  - Charge = 2.4 x (40)$^{1.05}$ $\simeq$ 116 mois-personne
  - Durée normale $\simeq$ 2.5 x (116)$^{0.38}$ $\simeq$ 12 mois
  - Taille moyenne de l'équipe = 116/12 = 9 personnes

Modèle Cocomo -suite-

Modèle intégrant des *facteurs correcteurs* (ou *d'ajustement*) liés au :

- logiciel : ratio données/programmes, complexité, ...
- matériel : disponibilité de certaines ressources pour les tests par exemple
- personnel : expérience, connaissances de l'environnement, ...
- projet : contraintes de délai, utilisation d'un IDE, ...

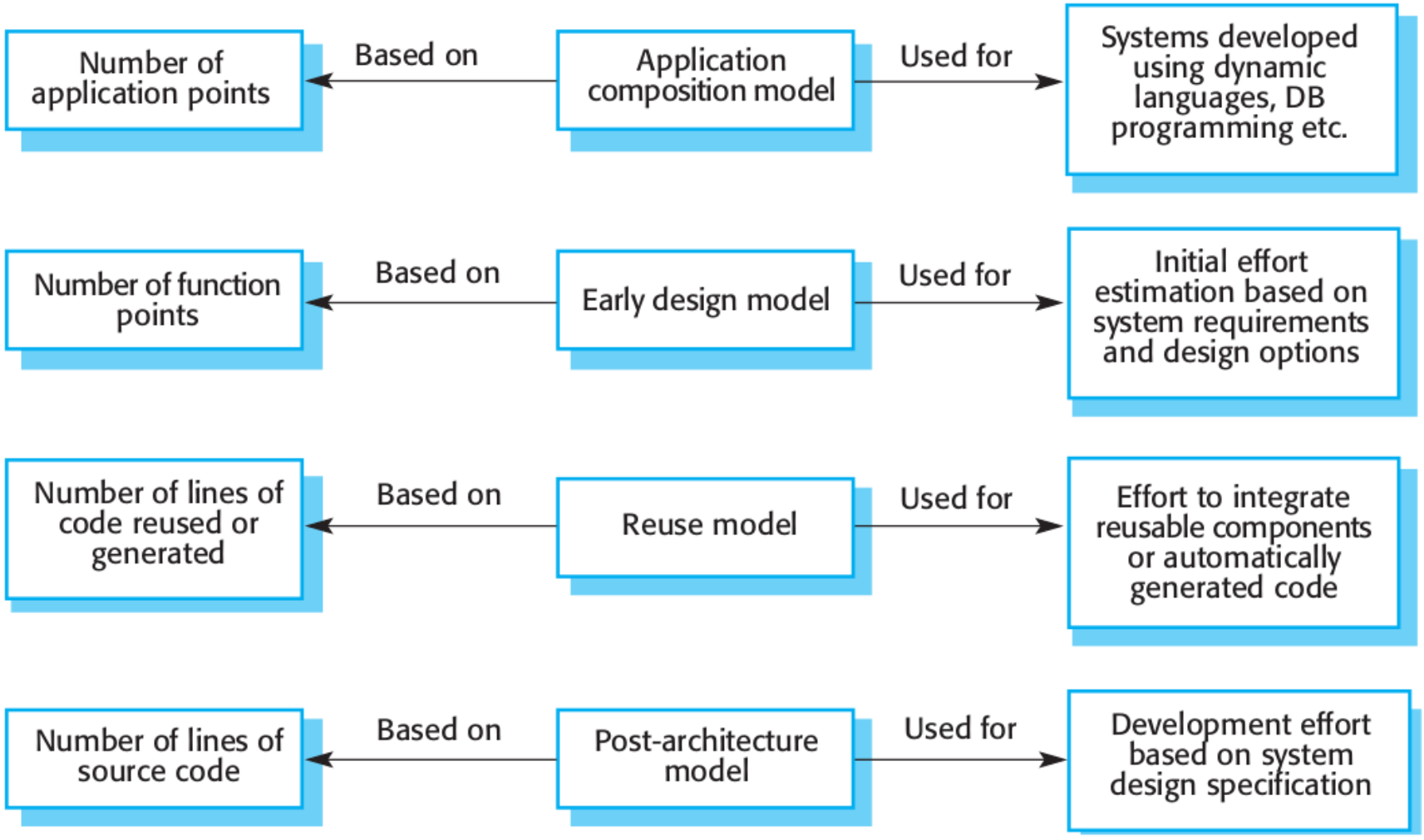
Charge nette = Produit des valeurs des facteurs correcteurs x charge brute

Modèle Cocomo II

- Cocomo II (1997) affine et actualise le modèle Cocomo
- Il donne des valeurs actualisées pour les facteurs correcteurs
- Il utilise les points de fonction (voir plus loin) en plus de SLOC comme unités d'oeuvre
- Il prend en compte les méthodes modernes de développement (RAD, réutilisation de code, génération automatique de code, ...)
- Outil en ligne (développé par la *Naval Postgraduate School, USA* ?) :

  `https://github.com/ekofebriharsono/COCOMOII`

Différents sous-modèles d'estimation



Sous-modèle de composition de l'application

- Estimer les charges d'un projet de développement avec réutilisation de modules (composition de l'application par assemblage de modules)
- Estimation basée sur des "points d'application" (nb de pages Web/écrans, de rapports générés, de modules réutilisés, de lignes de scripts, ...)
- Estimation ajustée avec des facteurs liés à la difficulté de dév des points d'app. (expérience des développeurs, utilisation d'IDE sophistiqués, ...)

Sous-modèle de composition de l'application -suite-

- Formule pour calculer l'effort en mois-personne (PM) :
  PM = (NAP x (1 - %reuse / 100)) / PROD
- NAP : nb total de points d'application
- %reuse : une estimation de la quantité de code réutilisée
- PROD : facteur d'ajustement lié à la productivité (capacité et expérience du développeur et capacité et maturité des IDE utilisés)
  Très faible = 4, Faible = 7, Nominale = 13,
  Haute = 25, Très haute = 50

Sous-modèle de conception précoce

- On suppose ici que les besoins ont été validés et que la conception est déjà entamée
- Estimation basée sur la formule du modèle Cocomo I : Effort = A x Taille^{B} x M
- On utilise ici le facteur d'ajustement (M) présenté précédemment

Sous-modèle de réutilisation

- Estimer l'effort d'intégration d'un code réutilisé ou généré
- Deux types de code réutilisé : *black-* & *white-box* code
- *black-box* code : pas nécessaire de le comprendre ou le modifier (ex : code généré à partir de certains modèles UML ou widgets graphiques)
- L'effort de dév de ce code = 0, et sa taille = 0, dans le calcul
- *white-box* code : code à adapter pour l'intégrer au reste de l'application (nécessite une phase de compréhension)
- Estimation mesurée avec la même formule que pour le sous-modèle précédent

Sous-modèle de réutilisation -suite-

- Taille du code additionnel calculée comme suit :
  ESLOC = (ASLOC x (1-AT/100) x AAM)
  - ASLOC : estimation du nombre de lignes de code à changer dans les composants réutilisés
  - AT : Pourcentage du code réutilisé qui peut être modifié automatiquement
  - AAM (*Adaptation Adjustment Multiplier*) : reflète l'effort additionnel requis pour réutiliser les composants (pourcentages représentant l'effort à rechercher des composants, à les comprendre, ...)
- Effort = A x $ESLOC^{B}$ x M

Sous-modèle de post-conception architecturale

- Estimation faite lorsque l'architecture a été conçue (on a une idée plus précise de la décomposition du système et sa taille)
- Calcul fait avec la même formule que précédemment, mais avec la taille totale du code :
  - estimation du nombre de lignes de code total : SLOC
  - estimation de la taille du nouveau code : ESLOC
  - estimation de la taille du code susceptible de changer à cause des changements futurs dans les besoins (*requirements*)

Calcul détaillé de l'exposant B

- L'exposant B dans la formule est calculé en prenant en compte 5 facteurs :
  - ancienneté dans la gestion de ce type de projets dans l'équipe,
  - flexibilité dans le développement (client fortement/faiblement impliqué),
  - analyse de risques,
  - cohésion de l'équipe,
  - et maturité du processus de dév
- Chaque facteur est noté entre 0 (très haut) et 5 (très bas)
- On fait la somme, on divise par 100 et on ajoute 1.01

Calcul de la durée estimée

- Le calcul de la durée se fait avec la formule suivante :
  $TDEV = 3 \times (PM)^{(0.33+0.2x(B-1.01))}$
- Unité : mois
- PM : mesure de l'effort avec les sous-modèles précédents
- B : exposant lié à la complexité (slide précédent)
- Ex : B = 1.17 et PM = 60 mois-personne
  $TDEV = 3 \times (60)^{0.36} = 13$ mois

Composition de l'équipe-projet

- Il y a une relation complexe entre le nombre de personnes dans une équipe-projet, l'effort estimé pour réaliser le projet et la durée estimée
- Ex : Pour un effort de 60 mois-personne à réaliser en 13 mois, il ne faut pas 5 personnes (pourant 13x5=65) mais plutôt 6 selon le modèle Cocomo
- Ajouter des personnes dans une équipe-projet ⇒ productivité moindre (+ de com et + d'effort à écrire les interfaces entre les sous-systèmes développés par chacun)
- Usuellement, on a besoin de peu de personnes au démarrage du projet (ne pas prévoir beaucoup de monde)
- On atteint un pic lors des phases de prog et tests
- On a besoin de peu de personnes à la fin

4. Estimation paramétrique : Méthode des points de fonction

- Méthode proposée chez IBM en 1979, formalisée par l'IFPUG (*International Function Point Users Group*) en 1986, et devenue norme ISO en 2003
- Elle permet de faire une estimation à partir d'une description externe du futur système (ses "composants fonctionels")
- Elle propose trois degrés de complexité (faible, moyen, élevé)
- Type de fonction + degré $\longrightarrow$ nombre de "points"
- Total permet d'évaluer les charges de réalisation de tout un projet en "points de fonctions"

Avantages de la méthode

- Mesures effectuées du point de vue de l'utilisateur
- Méthode indépendante d'une technologie en particulier (n'est pas lié à un langage de prog, par ex)
- Surcoût de la méthode faible (des statistiques ont montré un surcoût égal à seulement 1% du développement)
- Méthode qui marche bien avec les use-cases UML ou avec les méthodes agiles (*user stories*)
- Méthode permettant d'estimer le coût d'un projet, la durée d'un projet, la taille de l'équipe du projet, la vélocité (nombre de points de fonction par heure), ... (pas mal de métriques)

Composants fonctionnels

Cinq types de composants fonctionnels :

- Les données :
  - GDI (Groupe de Données Internes) : données internes au périmètre d'estimation et logiquement liées (objet de gestion créées et mis à jour au sein du périmètre)
  - GDE (Groupe de Données Externes) : externes au périmètre, mais interrogées
    GDE ou GDI = {DE (Données Élémentaires) et SLD (Sous-ensemble Logiques de Données)}
- Les transactions :
  - ENT (Entrée) : fonctions d'entrée de données métier de l'utilisateur dans le périmètre
  - SOR (Sortie) : fonctions de consultation des données métiers
  - INT (Interrogation) : fonctions d'extraction de données, sans mise à jour

Complexité et poids des GDI et GDE

	1 à 19 DE	**20 à 50 DE**	**51 DE ou +**
1 SLD	Faible	Faible	Moyenne
2 à 5 SLD	Faible	Moyenne	Élevée
6 SLD ou +	Moyenne	Élevée	Élevée

GDI

Complexité	**Points**
Faible	7
Moyenne	10
Élevée	15

GDE

Complexité	**Points**
Faible	5
Moyenne	7
Élevée	10

Tableaux similaires pour ENT, SOR et INT, avec des valeurs différentes (Nombre de GDE/GDI au lieu de SLD)

`https://alvinalexander.com/FunctionPoints/FunctionPoints.shtml`

Ajustement de l'estimation (non-normalisée)

- Somme des points = PFB (nombre de Points de Fonctions Brut)
- Estimation à ajuster (nombre de points de Fonctions Ajusté) :
  PFA = FA x PFB
- Pour calculer FA, on doit mesurer DIT (Degré d'Influence Total) :
  DIT = $\sum_{i=1}^{14} D_i$, $D_i \in [0,5]$
  (14 caractéristiques générales d'un système : portabilité, ...)
  $\Rightarrow$ DIT $\in [0 , 70]$
- FA = 0.65 + DIT/100, FA $\in [0.65 , 1.35]$
  C'est une valeur d'ajustement = +/- 35%
- Inconvénient : les 14 caractéristiques pèsent 35% / 14 = 2.5%
  (poids faible)

Transformation des points en charges

- Coefficient de transformation dépend de l'environnement du projet
- Il est recommandé que chaque entreprise établisse sa base de projets
- Certains auteurs ont proposé des formules de transformation des points en nombre d'instructions de code source (dépendant de langages de prog)
- Estimations généralement retenues :
  - Fin d'étude préalable : 1 point de fonction = 3 jours, 2 jours pour un petit projet et 4 pour un grand projet
  - Fin d'étude détaillée : 1 à 2 jours

Use Case Points (UCP) [2]

- Méthode développée en 1993 chez Objectory Systems (IBM)
- Méthode adaptée aux logiciels à objets et à l'utilisation d'UML (*Use Cases* (UC) pour définir les besoins)
- Pour calculer l'UCP, la méthode s'appuie sur :
  - *Unadjusted Use Case Weight (UUCW)* : nombre et complexité des UC
  - *Unadjusted Actor Weight (UAW)* : nombre et complexité des acteurs
  - *Technical Complexity Factor (TCF)* : facteur d'ajustement technique
  - *Environmental Complexity Factor (ECF)* : FA environnemental

2. Murali Chemuturi, Software Estimation Best Practices, Tools and Techniques for Software Project Estimators, J.Ross Publishing, 2009, pages 84-87

Unadjusted Use Case Weight (UUCW)

- Chaque use case est classé en : **Simple**, **Moyen** ou **Complexe** en fonction du nombre de transactions qu'il contient

Classification du UC	**N° de transactions**	**Poids (*weight*)**
Simple	1 à 3 transactions	5
Moyen	4 à 7 transactions	10
Complexe	Au delà de 8 transactions	15

- UUCW = (Nombre de UC simples x 5) + (Nombre de UC moyens x 10) + (Nombre de UC complexes x 15)

Unadjusted Actor Weight (UAW)

- Chaque acteur est classé en : **Simple**, **Moyen** ou **Complexe** en fonction de son type

Classif. acteur	**Type d'acteur**	**Poids (*weight*)**
Simple	Un système externe fournissant une API bien définie	1
Moyen	Un système externe avec lequel le SI va interagir en utilisant des protocoles standards (TCP, Base de données, ...)	2
Complexe	Acteur humain utilisant une GUI	3

- UAW = Nombre d'acteurs simples + (Nombre d'acteurs moyens x 2) + (Nombre d'acteurs complexes x 3)

Technical Complexity Factor (TCF)

- Affecter un score de 0 (impact nul) à 5 (facteur essentiel) à chaque facteur technique dans une liste de 13 facteurs (table ci-dessous)
- Multiplier chaque score par le poids dans la table et faire la somme (pour obtenir TF)

Facteur	Description	Poids	Facteur	Description	Poids
T1	Distributed system	2.0	T8	Portability to other platforms	2.0
T2	Response time/perf. objectives	1.0	T9	System maintenance	1.0
T3	End-user efficiency	1.0	T10	Concurrent/parallel processing	1.0
T4	Internal processing complexity	1.0	T11	Security features	1.0
T5	Code reusability	1.0	T12	Access for third parties	1.0
T6	Easy to install	0.5	T13	End user training	1.0
T7	Easy to use	0.5			

- TCF = 0.6 + (TF/100)

Environmental Complexity Factor (ECF)

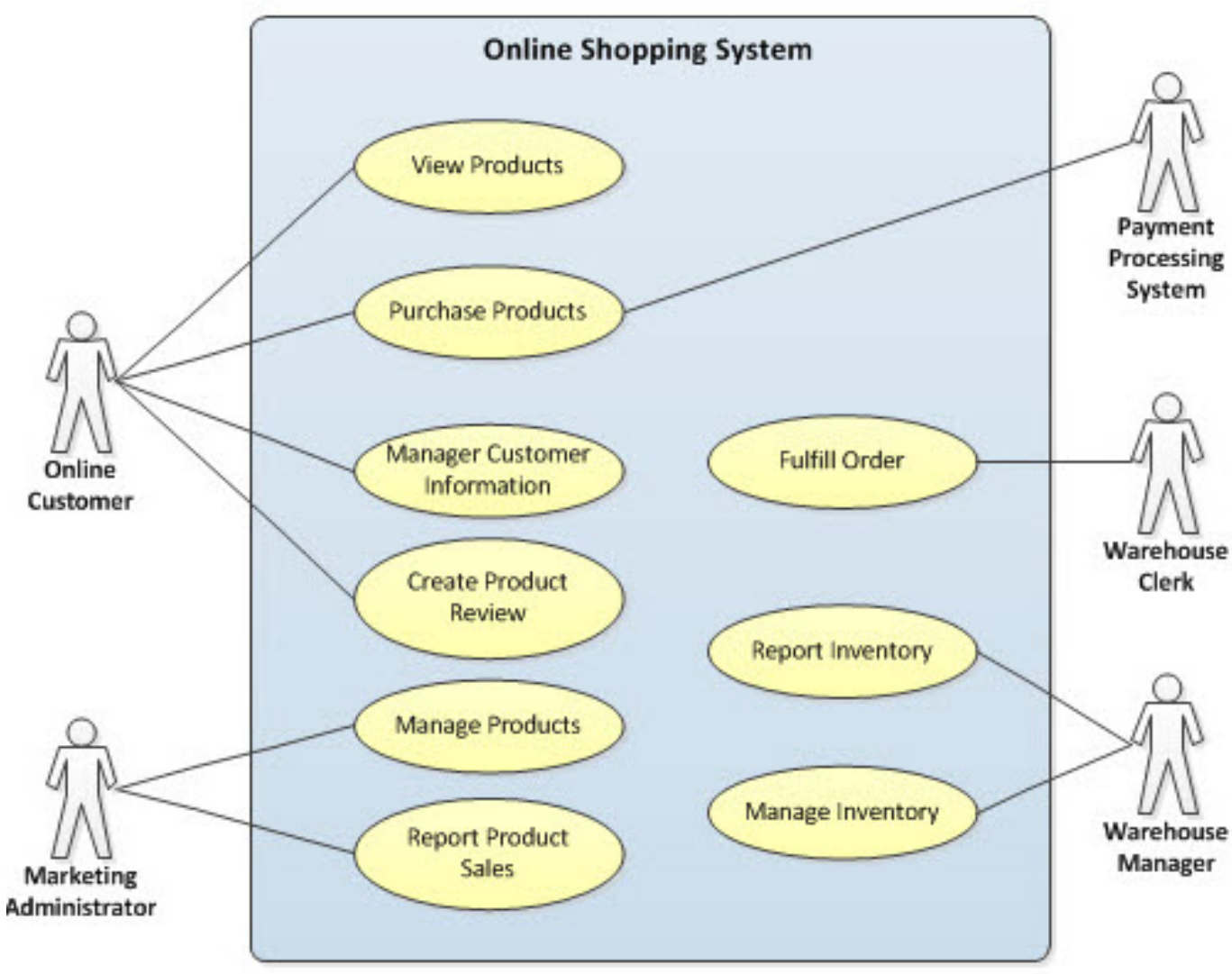
- Même principe que pour TF (pour obtenir EF)

Facteur	Description	Poids
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Object-oriented experience of team	1.0
E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	-1.0
E8	Difficult programming language	-1.0

- ECF = 1.4 + (-0.03 x EF)

UCP = (UUCW + UAW) x TCF x ECF

Exemple d'estimation avec UCP (source : Wikipedia)



Exemple d'estimation avec UCP : les mesures

- UUCW = (2 x 5) + (3 x 10) + (4 x 15) = 100
- UAW = (1 x 1) + (0 x 2) + (4 x 3) = 13
- TCF :

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
T1	Distributed system	2.0	5	10
T2	Response time/performance objectives	1.0	5	5
T3	End-user efficiency	1.0	3	3
T4	Internal processing complexity	1.0	2	2
T5	Code reusability	1.0	3	3
T6	Easy to install	0.5	1	0.5
T7	Easy to use	0.5	5	2.5
T8	Portability to other platforms	2.0	2	4
T9	System maintenance	1.0	2	2
T10	Concurrent/parallel processing	1.0	3	3
T11	Security features	1.0	5	5
T12	Access for third parties	1.0	1	1
T13	End user training	1.0	1	1
			Total (TF):	**42**

- TCF = 0.6 + (42/100) = 1.02

Exemple d'estimation avec UCP : les mesures

- ECF :

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
T1	Distributed system	2.0	5	10
T2	Response time/performance objectives	1.0	5	5
T3	End-user efficiency	1.0	3	3
T4	Internal processing complexity	1.0	2	2
T5	Code reusability	1.0	3	3
T6	Easy to install	0.5	1	0.5
T7	Easy to use	0.5	5	2.5
T8	Portability to other platforms	2.0	2	4
T9	System maintenance	1.0	2	2
T10	Concurrent/parallel processing	1.0	3	3
T11	Security features	1.0	5	5
T12	Access for third parties	1.0	1	1
T13	End user training	1.0	1	1
			Total (TF):	**42**

- ECF = 1.4 + (-0.03 * 10.5) = 1.085
- UCP = (100 + 13) x 1.02 x 1.085 = 125.06 (points UC)
- Exemple : 1 UCP = 28 heure-personne
- Effort estimé = 125.06 x 28 = 3501 heures ≃ 25 mois-personne

5. Estimation probabiliste

- Méthodes utilisées lorsque l'incertitude est importante (on s'appuie sur plusieurs valeurs d'estimation)
- Exemple : estimation à trois points et la méthode par simulation de Monte-Carlo

Estimation à trois points

Trois valeurs :

- valeur optimiste (f), hypothèse qu'on ne rencontrera aucun problème (dispo ressources, ...)
- valeur pessimiste (F)
- valeur la plus probable (p)

moyenne = (f + F + p) / 3

Parfois, on utilise une moyenne pondérée : moyenne = (f + F + 4p) / 6

5. Estimation probabiliste -suite-

Méthode par simulation de Monte-Carlo

- S'utilise généralement avec une méthode ascendante
- Méthode nécessitant d'avoir, pour chaque activité ou composant, une connaissance fine de la distribution des valeurs possibles de la charge (loi de probabilité)
- Simulation de Monte-Carlo : répéter n fois un tirage aléatoire des valeurs de la charge selon la loi de probabilité qui lui est associée
- On prend enfin la moyenne des valeurs obtenues