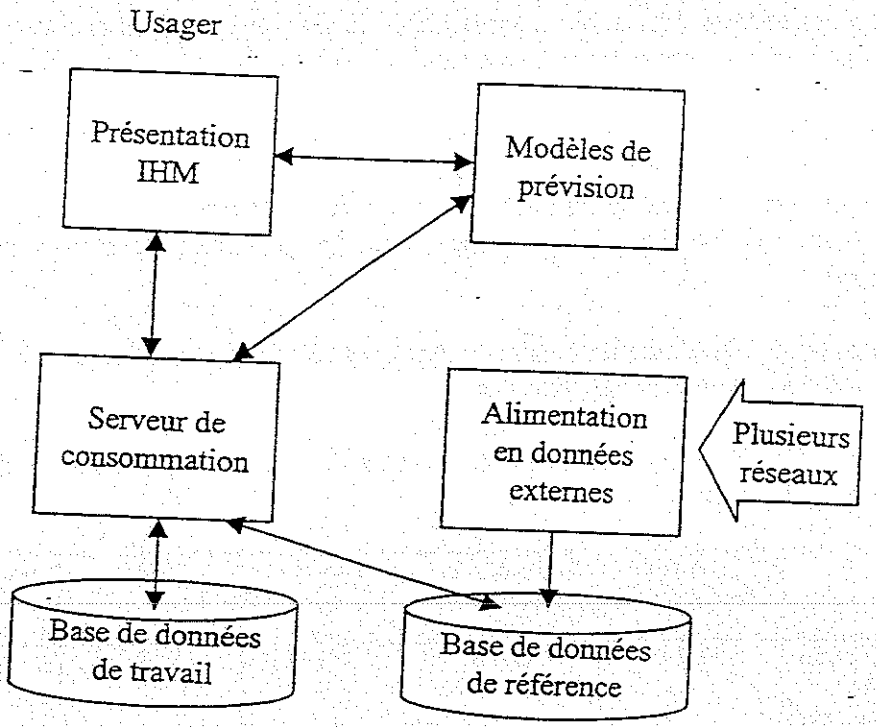
L'architecture de l'application, dans sa version industrielle a été réaménagée comme suit :



La base de données de travail qui ne sert que pendant une session utilisateur reste en objet ; par contre la base de données de référence qui doit conserver toutes les archives sur 20-30 ans est en relationnel.
Dans une première version de cette application, le matériel sera entièrement UNIX, puis, dans une seconde version, le poste de travail sera porté et adapté pour s'exécuter sur Windows-NT. Le module d'alimentation en données externes est considéré comme très critique.

Question N°1 : Reconstituer un profil de projet conforme au modèle COCOMO en faisant l'hypothèse que l'on peut assimiler le code source du prototype à du code de type S.

Question N°2 : Expliquer d'où provient la différence d'effort entre les données réelles du tableau ci-dessus et ce que vous venez de calculer à l'aide du modèle COCOMO. Pour cela, faite un tableau comparatif des efforts, comme suit :

	Conception	Programmation	Tests unitaires + Intégration
Prototype			
Modèle COCOMO			

Vous pourrez considérer que les 60 hm du prototype se répartissent en Conception = 5 hm, Programmation = 35 hm, TU et Intégration = 20 hm. Si vous n'êtes pas d'accord, expliquez pourquoi. Vous pourrez faire l'hypothèse que dans la phase de *Programmation tests unitaires*, les efforts sont répartis 50-50.

Question N°3 : On fait l'hypothèse qu'un projet réalisé avec une structure de coût conforme au modèle COCOMO conduit à un produit logiciel dont le taux d'erreurs résiduelles est estimé à 1.5

Err/KLS. De plus, on considère que la découverte des erreurs est proportionnelle à l'effort fourni, en particulier pendant la phase de tests, ce qui revient à dire que le taux d'erreurs résiduelles est inversement proportionnel à ce même effort. Enfin, on considérera, si nécessaire, que :

- à une ligne de conception correspond 10 lignes de code source (coefficient d'expansion de 10),
- le coefficient de propagation des erreurs de conception vers le code source est égal à 5 (ce qui veut dire que une erreur de conception produit 5 erreurs de programmation),
- 2/3 des erreurs résiduelles sont des erreurs de conception.

En prenant ces hypothèses en compte, calculer le nombre d'erreurs résiduelles probable du prototype.

Question N°4 : Quel risque prend la direction générale si elle décide de distribuer le prototype en l'état dans les 20 centres de prévision.

Question N°5 : Pour la réalisation de la version industrielle UNIX, les données du prototype sont corrigées comme suit :

	Serveur de consommation	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Total des lignes source intégrées
Lignes source avec commentaires	20.000	35.000	5.000	50.000	28.000	138.000
Lignes source sans commentaires	10.000 de type S	12.000 — dont 4.000 de type E, le reste de type S	3.000 de type S	25.000 de type P	7.000	57.000
Effort en hm					Non connu	

Les commentaires incorporés au code source constituent la documentation de conception détaillée.

Calculer l'effort et la durée nécessaire à ce nouveau projet.

Question N°6 : En supposant que l'on réutilise 80% du code développé pour le prototype, quel est l'effort à fournir pour passer du prototype à la version industrielle.

Question N°7 : Selon vous, où faut-il porter la priorité de l'effort de tests ? Proposer un pourcentage de répartition de cet effort sur chacun des modules.
Vous pouvez, par exemple calculer l'effort de test requis pour chacun des modules, puis répartir l'effort de test d'intégration global en fonction de ce qui vous paraît prioritaire.

NB : Vous pouvez présenter vos résultats comme suit :

	Serveur de consommation	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Conception globale et intégration finale
KLS	10	12	3	25	7	
% KLS						
Effort de test					Non connu	
% effort test					Non connu	

Question N°8 : L'exigence de portabilité Unix → Windows-NT de la partie *Présentation IHM* est-elle correctement prise en compte au niveau de la complexité du code source (25 KLS de type

P) ? On peut estimer que 10% du code est modifié par la portabilité. Comment faudrait-il structurer ce composant pour préparer correctement l'opération de portage ?

Cours B5 - Génie Logiciel

Corrigé de la question de cours

Question N°1 : Reconstituer un profil de projet conforme au modèle COCOMO en faisant l'hypothèse que l'on peut assimiler le code source du prototype à du code de type S.

	Serveur de consommation	Alimentation en données externes	Scripts (en langage de commandes)	IHM	Modèles de calcul (en Fortran)	Total des lignes source intégrées
Lignes source avec commentaires	17.500	19.400	4.300	84.700	28.000	~~151.900~~ 153.900
Lignes source sans commentaires	10.200	11.800	2.400	54.700	7.000	86.100
Effort en hm	18	10	2	30	Non connu	60

Seul les lignes sources sans commentaire sont à prendre en compte dans COCOMO. La différence constitue la documentation de ce programme, soit : 69.800 lignes de texte, soit à raison de 50 lignes par page, un document de 1.396 pages.

D'après ce tableau, l'application compte $86.1-7=79.1KLS \equiv 80KLS$ de code S.
Les 7KLS de code Fortran ne font pas parti du code à développer. On peut supposer qu'elles proviennent d'une bibliothèque de calcul.
Effort total selon COCOMO = $2.4(80)^{1.05}=239hm$ auquel il faut rajouter l'effort de management (6%) soit : $239\times1.06=253hm$.
Durée du développement = $2.5(239)^{0.38}=20mois$

Effort total selon Vade-mecum = $\frac{80}{0.35}=229hm$

Ventilation de l'effort :

Conception générale	Conception détaillée	Programmation et tests unitaires	Intégration
$0.16\times239=38hm$	$0.23\times239=55hm$	$0.36\times239=86hm$ Soit : ▪ 43 *hm* de programmation, ▪ 43 *hm* de tests unitaires	$0.25\times239=60hm$

Question N°2 : Expliquer d'où provient la différence d'effort entre les données réelles du tableau ci-dessus et ce que vous venez de calculer à l'aide du modèle COCOMO.
Vous pourrez considérer que les 60 hm du prototype se répartissent en Conception = 5 hm, Programmation = 35 hm, TU et Intégration = 20 hm. Si vous n'êtes pas d'accord, expliquez

6

pourquoi. Vous pourrez faire l'hypothèse que dans la phase de *Programmation tests unitaires*, les efforts sont répartis 50-50.
Pour cela, faite un tableau comparatif des efforts, comme suit :

	Conception	Programmation	Tests unitaires + Intégration
Prototype	5 *hm*	35 *hm*	20 *hm*
Modèle COCOMO	93 *hm*	43 *hm*	103 *hm*
Différence Mod-Proto	88 *hm*	8 *hm*	83 *hm*
Commentaires sur les différences	L'effort correspondant sert à organiser le système en modules et à définir les interfaces. Ceci est indispensable pour la maintenabilité du système, la fiabilité et le portage vers Windows-NT. Une partie importante de cet effort sert à valider la conception et découvrir des erreurs.	L'effort de programmation est presque identique. On peut dire que la différence sert à effectuer des relectures et des inspection de code qui permettront de découvrir quelques erreurs.	La différence sert à écrire des tests et à les exécuter. On peut considérer que dans le prototype il y a très peu de tests unitaires (pas de couverture, pas d'archivage) et quelques tests d'intégration. Il y aura beaucoup plus d'erreurs résiduelles dans le prototype.

Pour un prototype, et à fortiori pour une maquette, l'essentiel de l'effort va à la programmation car l'objectif n'est pas de faire un produit fini. Il est évidemment très dangereux de faire passer un prototype pour un produit logiciel qui, comme il est dit en cours, doit être parfaitement documenté et testé.

Question N°3 : On fait l'hypothèse qu'un projet réalisé avec une structure de coût conforme au modèle COCOMO conduit à un produit logiciel dont le taux d'erreurs résiduelles est estimé à 1.5 Err/KLS. De plus, on considère que la découverte des erreurs est proportionnelle à l'effort fourni, en particulier pendant la phase de tests, ce qui revient à dire que le taux d'erreurs résiduelles est inversement proportionnel à ce même effort. Enfin, on considérera, si nécessaire, que à une ligne de conception correspond 10 lignes de code source (coefficient d'expansion de 10) et que le coefficient de propagation des erreurs de conception vers le code source est égal à 5 (ce qui veut dire que une erreur de conception produit 5 erreurs de programmation).
En prenant ces hypothèses en compte, calculer le nombre d'erreurs résiduelles probable du prototype.

Le nombre d'erreurs selon le modèle est $1.5 \times 80 = 120$ erreurs.
Un premier calcul simpliste (purement proportionnel) permet d'établir le tableau de proportion suivant :
$93 + 103 = 196hm$ → 120 erreurs résiduelles.
$5 + 20 = 25hm$ → $120 \times \frac{196}{25} = 941$ erreurs résiduelles (c'est une valeur plancher).
Ce premier calcul suppose que l'effet de l'effort sur la qualité en terme d'erreurs résiduelle est le même en conception qu'en test, ce qui est une approximation grossière (Voir Que sais-je page 107).
Les 120 erreurs se répartissent en 80 erreurs de conception et 40 erreurs de programmation auxquelles il faut affecter des pondérations correspondant au efforts relatifs et à l'amplification, soit :

62

$80 \times \frac{93}{5} \times 5 + 40 \times \frac{103}{20} = 7440 + 200 = 7640$ erreurs résiduelles (c'est une valeur plafond, car comme il n'y a pas d'interface, le coefficient d'amplification est peut être un peu fort). Si on prend un coefficient d'amplification de 1, certainement trop faible, il y aura tout de même 1688 erreurs résiduelles. La vérité est entre ces deux valeurs. Avec une distribution de Pareto 80-20, on pourait avoir 80 erreurs avec un coef. 1 et 20 erreurs avec un coef. 5, ce qui conduirait à un coefficient d'amplification pondéré de 1.8, soit 2878 erreurs résiduelles probables.

Question N°4 : Quel risque prend la direction générale si elle décide de distribuer le prototype en l'état dans les 20 centres de prévision.

Le risque est considérable car le MTTF est une fonction décroissante du nombre d'erreurs résiduelles (elles sont au moins 20 fois plus nombreuses), et le MTTR dépend de la surveillance en ligne résultant de la conception, or il n'y a pas eu de conception.

Question N°5 : Pour la réalisation de la version industrielle UNIX, les données du prototype sont corrigées comme suit :

	Serveur de consommation	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Total des lignes source intégrées
Lignes source avec commentaires	20.000	35.000	5.000	50.000	28.000	138.000
Lignes source sans commentaires	10.000 de type S	12.000 dont 4.000 de type E, le reste de type S	3.000 de type S	25.000 de type P	7.000	57.000
Effort en hm					Non connu	

Les commentaires incorporés au code source constituent la documentation de conception détaillée.

Calculer l'effort et la durée nécessaire à ce nouveau projet.

Il faut effectuer une moyenne pondérée en fonction de la complexité du code, soit :
Type S 21 KLS → 0.42
Type P 25 KLS → 0.50
Type E 4 KLS → 0.08
D'où l'effort $0.42 \times 2.4(50)^{1.05} + 0.50 \times 3.0(50)^{1.12} + 0.08 \times 3.6(50)^{1.20} = 212hm$

Question N°6 : En supposant que l'on réutilise 80% du code développé pour le prototype, quel est l'effort à fournir pour passer du prototype à la version industrielle.

Ceci correspond à l'effort de programmation, soit $0.8 \times 43 = 35hm$.
L'effort supplémentaire à fournir est de : $212 - 35 = 177hm$.

Question N°7 : Selon vous, où faut-il porter la priorité de l'effort de tests ? Proposer un pourcentage de répartition de cet effort sur chacun des modules.
Vous pouvez, par exemple calculer l'effort de test requis pour chacun des modules, puis répartir l'effort de test d'intégration global en fonction de ce qui vous paraît prioritaire.

NB : Vous pouvez présenter vos résultats comme suit :

	Serveur de consommation –	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Conception globale et intégration finale
KLS	10	12	3	25	7	
% KLS						
Effort de test					Non connu	
% effort test					Non connu	

On peut calculer l'effort nécessaire à chacun des modules comme si il était seul, soit :

Serveur de consommation	$2.4(10)^{1.05} =$	27
Alimentation données externes	$0.67 \times 2.4(12)^{1.05} + 0.33 \times 3.6(12)^{1.20} =$	21,8 + 23,4 = 45,3
Script	$2.4(3)^{1.05} =$	7,6
IHM	$3.0(25)^{1.12} =$	110,4
Total :		

La différence avec les 212hm est à ventiler sur ce qui paraît être l'interface le plus prioritaire : l'alimentation en données externes, puis sur l'IHM.

Question N°8 : L'exigence de portabilité Unix → Windows-NT de la partie *Présentation IHM* est-elle correctement prise en compte au niveau de la complexité du code source (25 KLS de type P) ? On peut estimer que 10% du code est modifié par la portabilité. Comment faudrait-il structurer ce composant pour préparer correctement l'opération de portage ?

Un IHM avec générateur d'interface pourrait être assimilé à du code de type S. La partie purement dialogue avec l'usager est plutôt de type E (à cause des tests à effectuer avec différents usagers), et l'interface avec les données plutôt de type P. Les module impactés par la portabilité sont plutôt de type P, voire S selon la nature des interfaces systèmes utilisés.

64

EXAMEN DE GÉNIE LOGICIEL
Cours B5

- Deuxième Session 28 mars 1998
Locaux : ENSAM

La question de cours et l'exercice doivent être traités sur deux copies distinctes.

Prenez le temps de bien lire l'énoncé.

Tous documents et calculettes sont autorisés.

Barème :
Question de cours 12 points

Q1	1
Q2	3
Q3	2
Q4	2
Q5	2
Q6	2
	12

Q7	2
Q8	2
	4 (bonus)

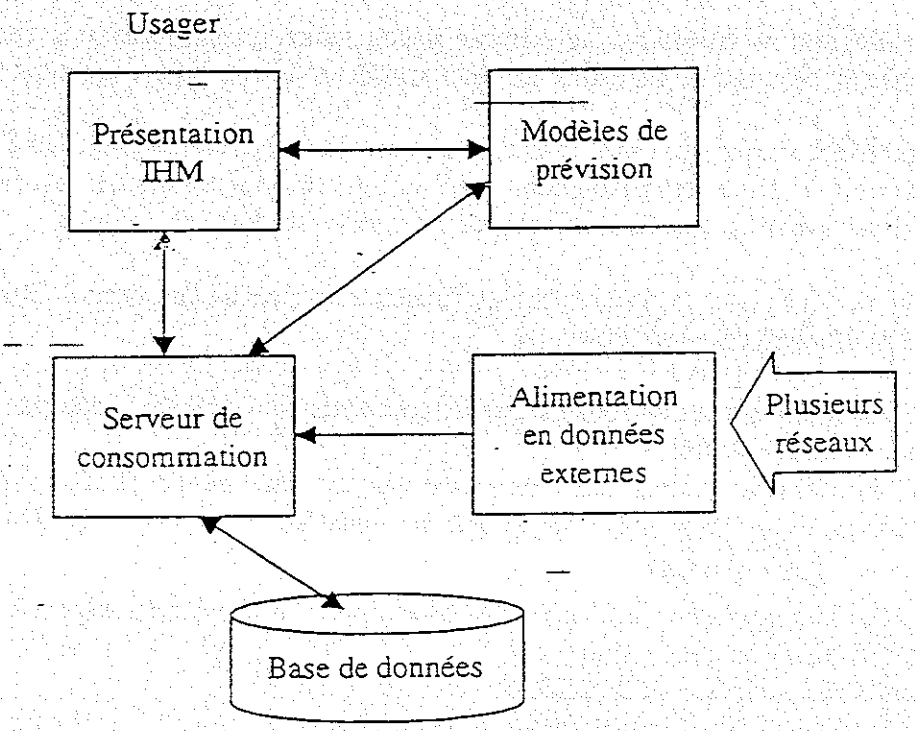
Cours B5 - Génie Logiciel

Question de cours

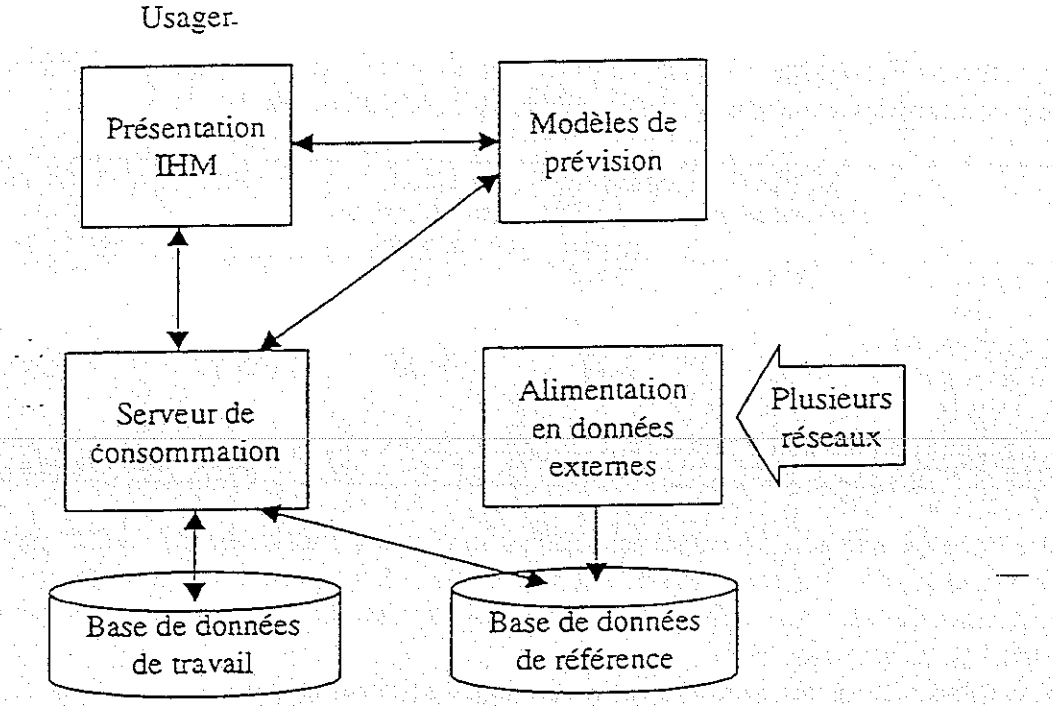
La société FranceElec a réalisé un prototype d'une application destinée à effectuer des prévisions de consommation électrique dont les données sont les suivantes :

	Serveur de consommation	Alimentation en données externes	Scripts (en langage de commandes)	IHM	Modèles de calcul (en Fortran)	Total des lignes source intégrées
Lignes source avec commentaires	17.500	19.400	4.300	64.700	28.000	133.900
Lignes source sans commentaires	10.200	11.800	2.400	54.700	17.000	96.100
Effort en hm	18	10	2	30	Non connu	60

Le code des modèles de calcul est réutilisé sans aucune modification ; il s'intègre dynamiquement à l'application lorsque celle-ci s'exécute. Le prototype a été réalisé selon une méthode incrémentale telle que celle de la méthode RAD (*Rapid Application Development*). La base de données est une base objet.
L'architecture de ce prototype est conforme au schéma suivant :



Après examen par la direction générale, il a été décidé de transformer ce prototype en application industrielle destinée à équiper 20 centres de prévision qui auront chacun en moyenne 10 copies de l'application, soit 200 systèmes installés en fin de déploiement.
L'architecture de l'application, dans sa version industrielle a été réaménagée comme suit :



La base de données de travail qui ne sert que pendant une session utilisateur reste en objet ; par contre la base de données de référence qui doit conserver toutes les archives sur 20-30 ans est en relationnel.
Dans une première version de cette application, le matériel sera entièrement UNIX, puis, dans une seconde version, le poste de travail sera porté et adapté pour s'exécuter sur Windows-NT. Le module d'alimentation en données externes est considéré comme très critique.

Question N°1 : Quel est le volume de documentation réellement écrit ? Selon votre propre expérience, quel effort cela représente-t-il ?

Question N°2 : Reconstituer un profil de projet conforme au modèle du vade-mecum.
NB : Rappel du vade-mecum

350 instructions source sans commentaire est une bonne productivité moyenne par personne et par mois pour un projet de complexité moyenne. La Durée moyenne est $=0.5\times\sqrt[3]{Effort(EnHA)}$.

☞ *Pour des applications à caractère technique et à dominante algorithmique (entre 500 et 1.000 KLS) on peut prendre :*
Programme simple : 4.000 LS/HA.
Programme à forte combinatoire : 2.000 LS/HA.
Programme de contrôle et/ou temps réel : 1.000 LS/HA.
Programme à tolérance de pannes et très fortes contraintes : 250 LS/HA .
Pour un programme composite il faut calculer une moyenne pondérée.

Question N°3 : Calculer la différence d'effort entre les données réelles du tableau ci-dessus et ce que vous venez de calculer à l'aide du modèle du vade-mecum. Pour cela, faites un tableau comparatif des efforts, comme suit :
Vous considérez que les 60 hm du prototype se répartissent en Conception = 5 hm, Programmation = 35 hm, TU et Intégration = 20 hm.

	Conception	Programmation	Tests unitaires + Intégration
Effort Prototype			
Effort Modèle vade-mecum			

Question N°3 : On considère que la différence provient du fait que l'on n'a corrigé que quelques défauts parmi tous ceux qui existent dans le code. En vous servant de la clé de répartition du vade-mecum (30-50-70, dans le §2 du vade-mecum), calculez l'effort qui correspond aux erreurs faites dans l'application.

Question N°4 : Calculer l'effort consacré à développer les tests. En faisant l'hypothèse qu'il faut une effort de 1hj pour fabriquer un test, quel est le nombre de tests ? Par rapport au nombre de lignes source, cela vous paraît-il raisonnable ? Quel est le coût des corrections ?

Question N°5 : On fait l'hypothèse qu'un projet réalisé avec une structure de coût conforme au modèle du vade-mecum ou de COCOMO conduit à un produit logiciel dont le taux d'erreurs résiduelles est estimé à 1.5 Err/KLS. De plus, on considère que chaque test a permis de découvrir trois erreurs. En vous servant des données réelles, calculez le nombre d'erreurs résiduelles du prototype.

Question N°6 : Quel risque prend la direction générale si elle décide d'installer le prototype en l'état dans un des 20 centres de prévision ? La situation résultante est-elle contrôlable ?

Question N°7 : Pour préparer la réalisation de la version industrielle UNIX, les données du prototype sont corrigées comme suit :

	Serveur de consommation	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Total des lignes source intégrées
Lignes source avec commentaires	20.000	35.000	5.000	50.000	28.000	138.000
Lignes source sans commentaires	10.000 de type S	12.000 dont 4.000 de type E, le reste de type S	3.000 de type S	25.000 de type P	7.000	57.000
Effort en hm					Non connu	

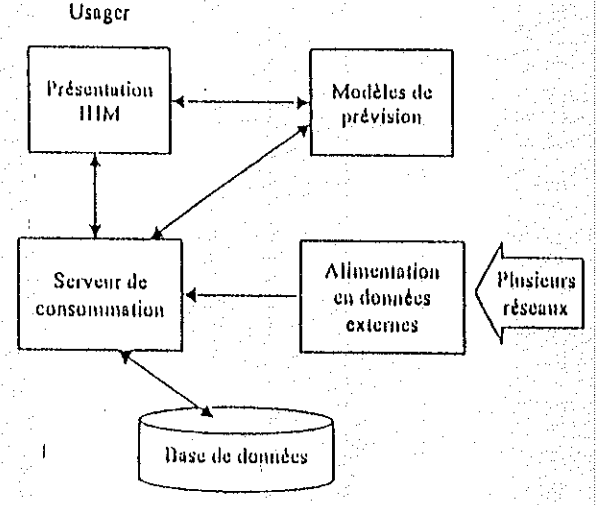
Les commentaires incorporés au code source constituent la documentation de conception détaillée.

Question de cours – CORRIGE

La société FranceElec a réalisé un prototype d'une application destinée à effectuer des prévisions de consommation électrique dont les données sont les suivantes :

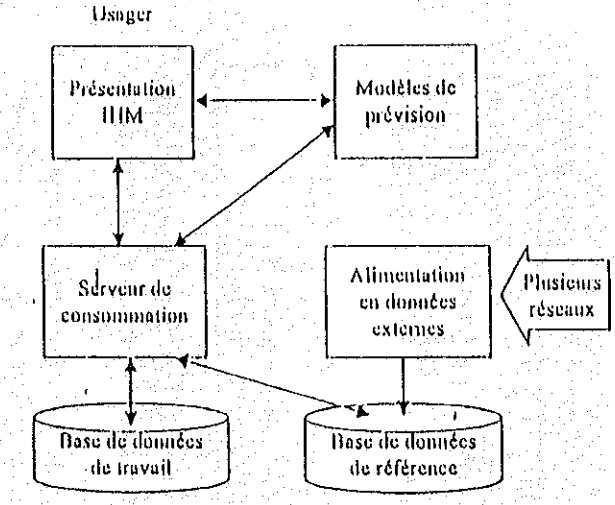
	Serveur de consommation	Alimentation en données externes	Scripts (en langage de commandes)	IHM	Modèles de calcul (en Fortran)	Total des lignes source intégrées
Lignes source avec commentaires	17.500	19.400	4.300	64.700	28.000	133.900
Lignes source sans commentaires	10.200	11.800	2.400	54.700	17.000	96.100
Effort en hm	18	10	2	30	Non connu	60

Le code des modèles de calcul est réutilisé sans aucune modification ; il s'intègre dynamiquement à l'application lorsque celle-ci s'exécute. Le prototype a été réalisé selon une méthode incrémentale telle que celle de la méthode RAD (*Rapid Application Development*). La base de données est une base objet.
L'architecture de ce prototype est conforme au schéma suivant :



Après examen par la direction générale, il a été décidé de transformer ce prototype en application industrielle destinée à équiper 20 centres de prévision qui auront chacun en moyenne 10 copies de l'application, soit 200 systèmes installés en fin de déploiement.

L'architecture de l'application, dans sa version industrielle a été réaménagée comme suit :



La base de données de travail qui ne sert que pendant une session utilisateur reste en objet ; par contre la base de données de référence qui doit conserver toutes les archives sur 20-30 ans est en relationnel.
Dans une première version de cette application, le matériel sera entièrement UNIX, puis, dans une seconde version, le poste de travail sera porté et adapté pour s'exécuter sur Windows-NT. Le module d'alimentation en données externes est considéré comme très critique.

Question N°1 : Quel est le volume de documentation réellement écrit ? Selon votre propre expérience, quel effort cela représente-t-il ?

Le code Fortran qui est réutilisé ne fait pas partie de cette application. Le volume de commentaires est donc :
$V = (17.500 + 19.400 + 4.300 + 64.700) - (10.200 + 11.800 + 2.400 + 54.700)$
$= 105.900 - 79.100 = 26.800$
Si l'on compte 50 lignes par page, cela fait un document de 536 pages.
Si l'on compte 10 pages/jours, cela fait 54 jours de travail, soit 2.7 hm auquel il faudra rajouter les cycles de relecture par différentes personnes et quelques inspections, soit 50% de plus. Au total, cette documentation aura nécessité un effort de $2.7 \times 1.5 = 4.05$hm, soit 6.75% de l'effort total pour le prototype.

Question N°2 : Reconstituer un profil de projet conforme au modèle du vade-mecum.
NB : Rappel du vade-mecum
350 instructions source sans commentaire est une bonne productivité moyenne par personne et par mois pour un projet de complexité moyenne. La Durée moyenne est $\approx 0.5 \times \sqrt{Effort(EnHJ)}$.

Pour des applications à caractère technique et à dominante algorithmique (entre 500 et 1.000 KLS) on peut prendre :
Programme simple : 4.000 LS/HA,
Programme à forte combinatoire : 2.000 LS/HA,
Programme de contrôle et/ou temps réel : 1.000 LS/HA,
Programme à tolérance de pannes et très fortes contraintes : 250 LS/HA .
Pour un programme composite il faut calculer une moyenne pondérée.

Le volume de code développé est de 79.100LS, soit : $\frac{79.100}{4.000} = 19.8ha$

La durée du développement est de $0.5 \times \sqrt{19.8} = 1.35\,an \cong 1\,an\,4mois$
Si on applique brutalement la règle 40-20-40, on a :
Conception (générale et détaillée) : 7.92 hm = 95 hm
Programmation et tests unitaires : 3.96 ha = 47.5 hm réparti 50-50
Intégration : 7.92 hm = 95 hm
La durée de la phase d'intégration est de 8 mois.

Question N°3 : Calculer la différence d'effort entre les données réelles du tableau ci-dessus et ce que vous venez de calculer à l'aide du modèle du vade-mecum. Pour cela, faites un tableau comparatif des efforts, comme suit :
Vous considérez que les 60 hm du prototype se répartissent en Conception = 5 hm, Programmation = 35 hm, TU et Intégration = 20 hm.

	Conception	Programmation	Tests unitaires + Intégration
Effort Prototype	5	35	20
Effort Modèle vade-mecum	95	47.5×0.5 = 23.5 hm	95 + 23.5 = 118.5 hm

Question N°3 : On considère que la différence provient du fait que l'on n'a corrigé que quelques défauts parmi tous ceux qui existent dans le code. En vous servant de la clé de répartition du vade-mecum (30-50-70, dans le §2 du vade-mecum), calculez l'effort qui correspond aux erreurs faites dans l'application.

Selon le vade-mecum :
30% de la conception est refaite et/ou modifiée, compte tenu des erreurs, soit : $0.3 \times 95 = 28.5hm$
50% de la programmation et TU est refaite et/ou modifiée, soit : 24hm
70% de l'effort de tests consiste à repasser les tests, soit : 66.5hm
L'effort de correction est donc de 119hm

Question N°4 : Calculer l'effort consacré à développer les tests. En faisant l'hypothèse qu'il faut une effort de 1*hj* pour fabriquer un test, quel est le nombre de tests ? Par rapport au nombre de lignes source, cela vous paraît-il raisonnable ? Quel est le coût des corrections ?

On s'intéresse essentiellement aux tests d'intégration. Selon le vade-mecum :

70% de l'effort d'intégration consiste à repasser les tests, 30% de l'effort a donc été consacré à écrire les tests d'intégration, soit : $0.3 \times 95 = 28.5hm = 627hj$ *en prenant des mois à 22 jours.*
Il y a donc 627 tests d'intégration pour 79.100 LS, soit une proportion de $\frac{79.100}{627} \cong 126LS$ *par test.*
Si l'on compte une instruction IF toutes les 8 ou 10 lignes, cela fait 12 à 15 conditions à vérifier. C'est probablement assez difficile compte tenu de ce qui a été dit dans le cours sur les graphes de contrôles et les couvertures. Avec ce jeu de tests, on a une couverture qui ne peut certainement pas être meilleure que 70-80%.
Le coût des corrections (conception + programmation) est de 28.5+24=52.5hm
On pourrait appliquer le même raisonnement à la partie TU, ce qui serait probablement un peu abusif sauf si on souhaite récupérer les TU en complément des tests d'intégration.

Question N°5 : On fait l'hypothèse qu'un projet réalisé avec une structure de coût conforme au modèle du vade-mecum ou de COCOMO conduit à un produit logiciel dont le taux d'erreurs résiduelles est estimé à 2 Err/KLS. De plus, on considère que chaque test a permis de découvrir trois erreurs. En vous servant des données réelles, calculez le nombre d'erreurs résiduelles du prototype.

Si chaque test découvre 3 erreurs, on a découvert $627 \times 3 = 1881$ *erreurs auxquelles il faut rajouter* $79.1 \times 2 = 158$ *erreurs, soit 2039 erreurs.*
Les données réelles indiquent que très peu de tests ont été réalisés, soit $0.3 \times 20 = 6hm = 132hj$ *soit 132 tests, si l'on applique les mêmes règles. Soit 396 erreurs découvertes, mais qui n'ont certainement pas été toutes corrigées compte tenu du très faible effort de correction qui résulte des données réelles.*
Dans le meilleur des cas où tout aurait été corrigé, il reste 2039-396=1643 erreurs, soit une densité de $\frac{1643}{79.1} = 21\,Err/KLS$*, soit 10 fois la densité du produit, ce qui n'est pas scandaleux compte tenu du fait qu'il s'agit d'un prototype.*

Question N°6 : Quel risque prend la direction générale si elle décide d'installer le prototype en l'état dans un des 20 centres de prévision ? La situation résultante est-elle contrôlable ?

Le site en question va servir de banc de test en situation réelle. Il sera très perturbé. Mais cela peut être un compromis acceptable si il y a du personnel de disponible dans le centre de prévision alors qu'il n'y en aurait pas dans le centre de développement du prototype. De plus le responsable du centre n'est pas obligé d'installer le prototype sur toute ses machines.
La situation est donc contrôlable.
Il serait évidemment catastrophique d'installer le prototype sur plus de un centre.

Question N°7 : Pour préparer la réalisation de la version industrielle UNIX, les données du prototype sont corrigées comme suit :

	Serveur de consommation	Alimentation en données externes	Scripts	IHM	Modèles de calcul	Total des lignes source intégrées
Lignes source	20.000	35.000	5.000	50.000	28.000	138.000

avec commentaires						
Lignes source sans commentaires	10.000 de type S	12.000 dont 4.000 de type E, le reste de type S	3.000 de type S	25.000 de type P	7.000	57.000
Effort en hjj					Non connu	

Les commentaires incorporés au code source constituent la documentation de conception détaillée.

Calculez selon le vade-mecum l'effort et la durée nécessaire à ce nouveau projet.

Il faut évidemment raisonner sur les LS sans les commentaires. L'effort doit se calculer par pondération, mais en première approximation on peut raisonner comme si tout était de type P (NB : calculer la pondération pour voir la différence).

Effort : $\frac{50.000}{2.000} = 25ha$

Durée : $0.5 \times \sqrt{25} = 1.46an$

	Conception	*Programmation*	*Tests unitaires + Intégration*
Effort par phase	*10 ha*	*5 ha×0.5= 2.5 ha*	*10 ha + 2.5 ha = 12.5 ha*

Question N°8 : Dans cet effort, il est prévu de passer tout le code et les tests en gestion de configuration. En faisant l'hypothèse qu'un module de code source fait 100 LS en moyenne, combien y a-t-il d'entités élémentaires à rentrer dans la base de données de la gestion de configuration ?

La partie programme de la base de configuration est organisée en une hiérarchie où les entités sont regroupées par paquets de 10 entités, jusqu'à arriver à l'entité qui contient les 50KLS de l'application. Combien y a-t-il d'entités intermédiaires dans la hiérarchie ?

Sachant qu'il faut 0.25 hj pour créer une entité dans la base de données de configuration, quel est l'effort pour mettre tout le produit sous gestion de configuration ?

Le nombre de modules de code est 50.000/100 = 500 entités élémentaires.
En appliquant la règle hiérarchique, il y a 50 entités de niveau 1 qui donnent naissance à 5 entités de niveau 2 et 1 entité application globale. Il y a donc 56 entités intermédiaires.
Calcul du nombre de tests :
On considère que les TU sont conçus comme les tests d'intégration.
Effort pour faire les tests d'intégration : $0.3 \times 10ha \times 220j = 660hjours$
Effort pour faire les TU : $0.5 \times 2.5ha \times 220j = 275hjours$
A raison de 1J par test on a 935 tests.
La base de données de gestion de configuration contient 500 + 56 + 935 = 1491 articles de configuration ; soit un effort de création de la base : 1491×0.25 = 373hj =1.7ha

EXAMEN DE GÉNIE LOGICIEL
Cours B5

Session I 13 mars 1999
Arcueil

La question de cours et l'exercice doivent impérativement être traités sur deux copies distinctes.

Prenez le temps de bien lire l'énoncé.

Tous documents et calculettes sont autorisés.

Barème :
Question de cours 12 points
- Q1 4
- Q2 3
- Q3 3
- Q4 2

Bonus
- Q5 2
- Q6 2

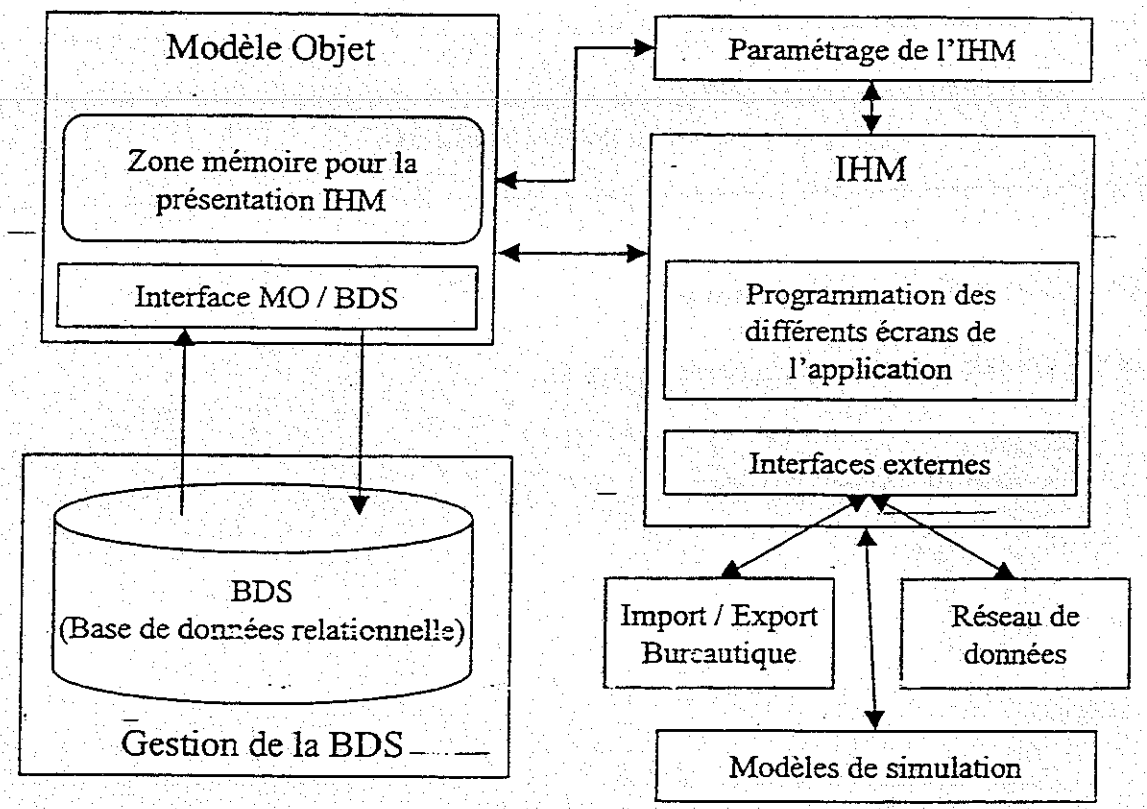
72

Cours B5 - Contrôle de Génie Logiciel

Etude de cas

La société Power Inc. qui agit comme maître d'ouvrage souhaite industrialiser un prototype d'une application destinée à effectuer des prévisions de consommation électrique. L'application est considérée comme vitale du point de vue économique, car son bon fonctionnement évitera des investissements en équipements lourds très coûteux (plusieurs milliards de F). La duré de vie prévisible de l'application est de 15-20 ans.

L'architecture de l'application est donnée par le schéma suivant :



L'IHM (i.e. Interface Homme Machine) et le Modèle Objet forment un bloc fonctionnel IHM/MO constituant le lot de réalisation N° 1. Le modèle objet est un programme de gestion mémoire destiné à préparer les données nécessaires aux différents écrans utilisés par l'opérateur et aux résultats des modèles de calcul en FORTRAN utilisés pour la simulation. L'application est complètement programmée en C++. Le SGBD de la base de données BDS (Base de Données de Simulation) est ORACLE. A terme, le bloc fonctionnel IHM/MO constituera un poste client, le bloc BDS constituera le serveur de l'application auquel plusieurs clients pourront se connecter. Le maître d'ouvrage Power Inc. a sélectionné un maître d'œuvre industriel pour effectuer les travaux de développement et d'intégration de l'application complète.

Les données de la gestion de projet (estimation initiale) fournies par le maître d'œuvre industriel sont les suivantes :

	Charges *en h.j*	Descriptif du lot
Evaluation Lot 1 (*1), dont :	692	Le lot 1 correspond à l'industrialisation du prototype de l'IHM réalisé sur station UNIX. La version industrielle est réalisée sous Windows NT. Elle est entièrement reprogrammée par une nouvelle équipe d'ingénieurs débutants ; seul le responsable de l'équipe qui est également le chef de projet a participé à la réalisation du prototype.
Conception :	42	
Codage :	360	
Test :	75	
Recette et corrections :	50	
Coordination et AQ :	52	
Evaluation Lot 2, dont :	441	Le lot 2 correspond à l'industrialisation de la base de données de simulation (BDS). La conception de cette base est identique à celle du prototype (le modèle conceptuel est le même). Seule l'interface avec l'IHM est nouveau. La version industrielle doit prendre en compte des contraintes de fiabilité, de performance et de maintenabilité.
Conception :	45	
Codage :	279	
Test :	...	
Recette et corrections :	45	
Coordination et AQ :	16	
Evaluation Lot 3, dont :	69	Le lot 3 est l'intégration de l'IHM/MO et de la BDS
Conception :	3	
Codage :	50	
Test :	3	
Recette et corrections :	3	
Coordination et AQ :	4	
Evaluation Lot 4, dont :	380	Le lot 4 réalise un complément d'IHM, en particulier en terme de paramétrage, d'aides en ligne et de messages d'erreurs.
Conception :	15	
Codage :	205	
Test :	45	
Recette et corrections :	30	
Coordination et AQ :	30	
Evaluation Lot 5, dont :	263	Le lot 5 intègre de nouvelles fonctions de simulation.
Conception :	12	
Codage :	130	
Test :	33	
Recette et corrections :	20	
Coordination et AQ :	18	
Evaluation Lot 6, dont :	220	Le lot 6 réalise des fonctions d'export des résultats de simulation vers d'autres applications et vers la bureautique pour les éditions.
Conception :	3	
Codage :	160	
Test :	16	
Recette et corrections :	...	
Coordination et AQ :	18	
Evaluation Lot 7, dont :	247	Le lot 7 réalise des fonctions de mise en librairies de façon à gérer les contextes et les données de travail nécessaires à la simulation, ainsi que des sauvegardes.
Conception :	50	
Codage :	135	
Test :	...	
Recette et corrections :	30	
Coordination et AQ :	7	

Evaluation Lot 8, dont : Conception : Codage : Test : Recette et corrections : Coordination et AQ :	75 ... ... 65 10	Le lot 8 est un lot de validation globale de l'application.
Evaluation Lot 9+10, dont : Conception : Codage : Test : Recette et corrections : Coordination et AQ :	34 1 15 3 10 2	Les lots 9 et 10 est un petit complément qui intègre une fonction nouvelle dans les librairies (c'est un complément du lot 7).
Total de l'estimation en jours	2421j	
Bilan des sous estimations		Donner les sous estimations par lot, là où vous penser qu'il y a eu sous-estimation.

(*1) : La différence entre l'évaluation totale du lot et le détail correspond à des postes comme le suivi de projet, la coordination avec la maîtrise d'ouvrage, la rédaction de la documentation et des divers spécifiques à chaque lot.

NB. : pour les conversions en homme.mois on prendra 1 mois = 21 jours ; pour les conversions en homme.an on prendra 1 an = 220 jours.

Question N°1 : En utilisant les modèles d'estimation présentés dans le cours (COCOMO ou le modèle rudimentaire du Vade mecum du chef de projet) quels sont les lots qui vous paraissent mal évalués, et de combien ? Qu'est ce qui permettrait d'expliquer ces décalages dans le contexte du projet (l'industrialisation a été précédée par un prototype). Dans tous les cas justifier brièvement vos analyses.
Quel serait selon vous le montant de l'effort à rajouter globalement pour retrouver un profil de charges plus convenable, en particulier pour ce qui concerne l'intégration de l'ensemble.
Est-il judicieux de procéder à une intégration intermédiaire (Cf. lot 3) ? justifier votre réponse.
Est-il logique de trouver de la programmation dans un lot d'intégration ? justifier votre réponse.
Le lot 3 vous paraît-il correctement estimé par rapport aux autres lots ?

Question N°2 : En utilisant les données projet du lot N°1 calculer le volume de programmation probable correspondant aux 360 jours qui ont été alloués à cette tâche. On conviendra que la tâche de programmation correspond à l'écriture de programmes qui sont compilés sans erreur. L'activité de tests unitaires est comptabilisée dans les rubriques Test et Recette/Corrections. Justifier et expliquer la productivité de la programmation en utilisant les données COCOMO ou le modèle rudimentaire du Vade mecum du chef de projet.
A défaut, en fonction de votre propre expérience, indiquez le volume de programmation (hors spécification, conception et test) que vous êtes capable de produire en 360 jours de travail normal, i.e. 8h par jour.

Question N°3 : Le volume de programmation estimé ci-dessus comporte un certain nombre d'instructions telles que : (IF, CASE OF, DO WHILE ou UNTIL, etc.) qui altèrent le flot de contrôle quand le programme s'exécute. Ces instructions permettent de construire le graphe de contrôle du programme. Pour simplifier les calculs, on considérera qu'il y a en moyenne une instruction de contrôle toutes les 10 instructions. Quel est le nombre moyen d'instructions de contrôle ?

Sachant qu'une instruction de contrôle contient une expression conditionnelle qui manipule en moyenne 4 variables (par exemple dans des expressions comme IF A(I)=B(J) ...), combien faudrait-il écrire d'instructions d'assignation pour faire varier toute les conditions au moins une fois dans le cas vrai et dans le cas faux ? Expliquez vos hypothèses de calcul.
NB : pour une condition telle que A(I)=B(J), il faut écrire une séquence comme :

I=*v1*
A(I)=*v2*
J=*v3*
A(J)=*v4*

v1, v2, v3, v4 étant des valeurs judicieusement choisies conformes aux types de ces différentes variables.
Un tel programme constitue un programme de test associé au programme initial qui lui a donné naissance (c'est d'ailleurs ce que vous écrivez quand vous utilisez un debugger symbolique pour forcer les variables de vos programmes !). Le texte obtenu est structuré en bloc de 100 lignes (i.e. une double page) en moyenne constituant 1 test au sens habituel du terme.
Combien obtient-on de tests avec une telle méthode de construction des tests ?

Question N°4 : Le nombre de tests calculé ci-dessus est jugé suffisant pour amener le programme à un niveau de qualité satisfaisant.
Si l'on considère qu'en fin de programmation (au sens précisé ci-dessus) il y a en moyenne 15 erreurs par millier de lignes source, le nombre de tests vous paraît-il suffisant ? Justifier votre réponse.

Question N°5 : Combien de fois faut-il exécuter les tests pour assurer la non régression du programme à tester suite aux modifications effectuées sur ce programme.
Si les résultats des tests doivent être dépouillés et analysés manuellement, quel est l'effort correspondant (prendre une durée de 1 heure pour un dépouillement de 1 tests, ou une autre valeur qui vous paraîtrait plus raisonnable mais que vous justifierez) à un passage complet de tous les tests avec une non régression.
NB. Le nombre de passage est une somme S=1+2+3+...+n, n étant le nombre de tests disponibles.

Question N°6 : A partir du programme de tests élaboré à la question 3, que faudrait t-il faire pour rendre le dépouillement des résultats automatique (NB. : ce que vous vérifiez à l'œil nu lors du debugging du programme peut être également programmé, ce qui augmentera d'autant le volume du programme de test). Estimer l'accroissement de taille du programme de test.
Selon vous quel est le choix le plus raisonnable, dépouiller manuellement, ou rendre le programme de test automatique ? Expliquer et justifier votre réponse.

Evaluation Lot 8, dont : Conception : Codage : Test : Recette et corrections : Coordination et AQ :	75 ... ... 65 10	Le lot 8 est un lot de validation globale de l'application.
Evaluation Lot 9+10, dont : Conception : Codage : Test : Recette et corrections : Coordination et AQ :	34 1 15 3 10 2	Les lots 9 et 10 est un petit complément qui intègre une fonction nouvelle dans les librairies (c'est un complément du lot 7).
Total de l'estimation en jours	2421j	
Bilan des sous estimations		Donner les sous estimations par lot, là où vous pensez qu'il y a eu sous-estimation.

(*1) : La différence entre l'évaluation totale du lot et le détail correspond à des postes comme le suivi de projet, la coordination avec la maîtrise d'ouvrage, la rédaction de la documentation et des divers spécifiques à chaque lot.

NB. : pour les conversions en homme.mois on prendra 1 mois = 21 jours ; pour les conversions en homme.an on prendra 1 an = 220 jours.

Première partie

Question N°1 : En utilisant les modèles d'estimation présentés dans le cours (COCOMO ou le modèle rudimentaire du Vade mecum du chef de projet) quels sont les lots qui vous paraissent décalés, et de combien ? Qu'est ce qui permettrait d'expliquer ces décalages dans le contexte du projet (l'industrialisation a été précédée par un prototype). Dans tous les cas justifier brièvement vos analyses.
Quel serait selon vous le montant de l'effort à rajouter globalement pour retrouver un profil de charges plus convenable, en particulier pour ce qui concerne l'intégration de l'ensemble.
Est-il judicieux de procéder à une intégration intermédiaire ? justifier votre réponse.

Réponse : Analyse des charges des différent lots

L'analyse des charges fait apparaître des clés de répartition qui ne sont pas conformes, et de très loin, à l'état de l'art. En particulier, le poste TEST est très fortement sous-estimé, à supposer que la charge de codage est correcte. De plus, le poste INTEGRATION est virtuellement absent.
Pour information, voici une réestimation avec des données que j'ai pu consolider au moyen de nombreuses expertises en matière de CQFD.

Hypothèses de calcul :

Statistiquement, on observe presque toujours une répartition voisine de 40-20-40 entre Conception, Codage/tests unitaires, Intégration. Dans ce projet, on peut faire l'hypothèse que la conception résulte des prototypes : donc pas de coefficient correcteur bien que les chiffres du lotissement soient très éloignés du modèle nominal.