

## COBOL cours 9

- Règles d'écriture
- Logique traitement de fichier Cobol
- Exercice
- Réflexion sur le métier de l'informatique.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

1

## Règles d'écritures de programmes clairs

- Les règles qui suivent sont extraites de l'article Cobol Under Control de Henry F. Ledgard et William C. Cave paru dans Communications of the ACM, novembre 1976, volume 19, numéro 11.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

2

## Règles Générales.

- GE-1 : Toute violation du présent standard doit être approuvée par le chef de projet.
- GE-2 : Pour chaque installation et pour chaque application, il doit être convenu d'un ensemble d'identificateurs communs et standard.
- GE-3 : Chaque installation doit définir des conventions de présentation des programmes (indentations).
- GE-4 : Les parties logiques d'un identificateur doivent être séparées par un tiret.
- GE-5 : Seules les colonnes 8 à 72 doivent être utilisées par le source des instructions.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

3

## Règles relatives à l'IDENTIFICATION DIVISION

- ID-1 : Le paragraphe AUTHOR de chaque programme en cours de réalisation doit contenir le nom de toutes les personnes qui ont écrit ou modifié une partie du programme.
- ID-2 : L'IDENTIFICATION DIVISION doit être suivie par de lignes de commentaires donnant un bref résumé des fonctions des programmes et les références externes intéressantes.
- ID-3 : Quand un programme est en activité, chaque modification de celui-ci doit faire l'objet d'une description avant chaque recompilation

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

4

## Règles relatives à la DATA DIVISION

- DA-1 : Toutes les spécifications des données externes susceptibles d'être utilisées par un programme doivent être disponibles dans une bibliothèque accessible à tous.
- DA-2 : Les variables structurées doivent utiliser les numéros 01, 02, 03, ..., sans aucun saut dans la numérotation.
- DA-3 : Les entrées 77 sont interdites.
- DA-4 : En dehors des images décrivant des données éditées, les clauses PICTURE ne doivent pas contenir des séquences de deux ou plus symboles identiques.
- DA-5 : Toutes les données qui restent constantes au cours de l'exécution du programme doivent être initialisées avec la clause VALUE et non dans la PROCEDURE DIVISION. Inversement, une donnée initialisée avec la clause VALUE ne doit pas être modifiée par le programme.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

5

## Règles relatives à la PROCEDURE DIVISION 1/2

- PR-1 : Le nombre maximum de lignes dans un paragraphe est celui imprimable sur un feuillet (*sur un écran*).
- PR-2 : Les instructions GO TO sont interdites.
- PR-3 : L'instruction STOP RUN ne doit être présente que comme dernière instruction du paragraphe principal. L'instruction EXIT ne doit intervenir que comme dernière instruction du paragraphe principal d'un sous-programme.
- PR-4 : L'imbrication des structures de séquençement IF ne doit pas excéder trois niveaux.
- PR-5 : L'instruction IF doit être utilisée pour simuler le **case** du Pascal.
- PR-6 : L'option THRU est interdite dans un PERFORM.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupeuz

6

## Règles relatives à la PROCEDURE DIVISION 2/2

- PR-7 : La valeur des variables associées à l'option VARYING du PERFORM ne doit pas être modifiée à l'intérieur de la boucle.
- PR-8 : Les opérations d'entrées-sorties doivent être isolées du reste du programme.
- PR-9 : S'il y a un risque qu'une donnée lue depuis l'extérieur soit incorrecte alors elle doit être contrôlée dès sa lecture.
- PR-10 : Tous les calculs arithmétiques doivent utiliser le verbe COMPUTE. ADD et SUBTRACT sont autorisées pour les incréments et décréments. DIVIDE est autorisé pour calculer un reste.
- PR-11 : Les parenthèses doivent être utilisées pour préciser l'ordre d'évaluation d'une condition complexe.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

7

## Programme COBOL

- Structure des données en entrées
- Structure des données en sorties
- Structure des traitements
- Fusion des structures.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

8

## Exemple 1

- Supposons que l'on dispose d'un fichier du personnel trié par services, et possédant un enregistrement pour chaque employé de l'entreprise. On veut obtenir un état sur lequel figure pour chaque service le nombre d'employés en faisant partie.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

9

## Exercice 1 1/2

- **But du programme**
- Calculer la prime de fin d'année de chaque employé en fonction de son ancienneté et de sa catégorie.
- **Fichiers**
- **EN ENTREE :**
- Le fichier des employés dont chaque enregistrement contient :
  - pos. 1 à 6 i le n° d'employé composé d'une position pour le code catégorie et de 5 positions pour le n° matricule.
  - pos. 7 à 30m nom et prénom.
  - pos. 31 à 38m date d'entrée dans l'entreprise, sous la forme JJMMAAAA.
- **EN SORTIE :**
- Une liste à l'imprimante reprenant pour chaque employé son n°, nom-prénom, date d'entrée et montant de la prime. (Voir dessin d'état).

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

10

## Exercice 1 2/2

- **Traitements.**
- 3.1. Traitement préliminaires.
- Lecture, au terminal clavier-écran, d'un paramètre indiquant l'année à laquelle les primes sont calculées. Cette année est donnée sous la forme 20XX.
- 3.2. Traitement enregistrement employé.
- Calculer la prime de fin d'année sur base des éléments suivants-
  - Si catégorie = 0
    - et si ancienneté inférieure à 1 pas de prime;
    - si ancienneté sup ou = à 1: 100€
    - si ancienneté supérieure à 10 : 500€ + 100 € par année d'ancienneté au delà de la dixième avec un maximum total de 1500 €.
  - Si catégorie <= 0
    - et si ancienneté inférieure à 1 : 80€
    - si ancienneté sup ou = à 1: 350 €
    - si ancienneté supérieure à 10 : 350 € + 80€ par année d'ancienneté au delà de la dixième avec un maximum total de 1000 €.

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

11

## Exercice 2

- Supposons que l'on veuille écrire le numéro de référence des factures d'une entreprise importante. Celle-ci utilise un numéro composé de trois lettres suivant le service émetteur, d'un numéro de facture pour le service sur cinq positions, d'un tiret, de l'année sur deux positions pour faciliter les recherches dans les archives (ex. : PRT00128-89).
- Reprenons l'exemple de notre entreprise et de ses factures. Elle veut obtenir en fin d'année un récapitulatif sur le chiffre d'affaire réalisé par chaque service et par mois.
- Le fichier d'entrée sera constitué de la liste des factures où chaque article sera une facture telle que nous l'avons défini la dernière fois.
- En sortie, la direction veut obtenir un tableau où les libellés des services concernés sont indiqués en clair, avec totaux par service et par mois, plus total général, de la forme suivante :
- 
- Services/Mois J F M .... D TOTAL
- Prêts
- Comptabilité
- Audits
- Stages
- Informatique
- TOTAUX

12/11/2002

COBOL Théorie 2002 / Pierre  
Renoupez

12

## Réflexion:

http://dchaffiol.free.fr/info/developpeur/art\_toujoursPareil\_1.htm

[« Mais si... 30 ans qu'on fait la même chose que ? \(vous dis...\) » \(1970/2000\)](#)

- Vous êtes jeune, vous êtes "in", branché, dans l'com, contempteur votre projet Java d'un oeil tout rapatriant du net votre dernier composant EJB/XML de l'autre et écoutant d'une troisième oreille votre lecteur de mp3.  
Mais...  
Mais il y a ce consultant senior, 30 ans de métier, qui vous regarde en rigolant depuis une bonne demi-heure... à force, ça veax.  
Alors vous n'y tenez plus, vous vous retournez et l'apostrophez. Et il commence à parler. Et vous arrêtez votre lecteur mp3.

Et si... et si vous regardez l'informatique du point de vue "métier" ? Qu'est-ce que cela veut dire ? Toutes les réponses sont dans ce témoignage.  
Ok, il est long... mais lisez-le tranquillement, vous verrez... il fait réfléchir, ce témoignage.

Mes "questions" sont en *italique rouge*  
Les renseignements essentiels de cet entretien sont *surlignés*

*30 ans de métier... et vous êtes diplômé en informatique ??? Dans les années (1970) ?*

Et comment, je l'ai dit. Deuxième promotion (d'un des trois premières écoles d'ingénieur en informatique de Paris, s'il vous plaît. Avant, il n'y en avait qu'une en France, à Grenoble. Nous étions donc en plein dans ce que l'on appelait à l'époque les ordinateurs de "troisième génération", celle qui a vu l'apparition des circuits intégrés, de la RAM et ROM, plus généralement l'avènement des super calculateurs, immenses monstres d'acier qui s'étalaient sur des pièces entières.

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

13

## Réflexion

- 1970... Comment la France se positionnait vis-à-vis de l'informatique ?

Bah, comme pour beaucoup d'autres sujets, en indépendante (surtout vis-à-vis des américains). De Gaulle (le général) avait lancé le "plan calcul" (le 18 juillet 1966) qui était là pour s'affranchir des constructeurs américains et aussi pour fournir des données pour le programme nucléaire naissant. L'IRIA (Institut de Recherche en Informatique et Automatique) émit créée par une Loi de novembre 1967. Le Pascal venait d'apparaître en 1969 et en 1970... le pacte français d'ordinateurs se montait à 4500 unités. La seule industrie française dans ce domaine technologique était Bull... dont la General Electric avait des participations importantes.

30 ans à faire la même chose... mais quelle chose ?

Mais l'informatique du côté métier. Vous, les jeunes, vous voyez toujours l'informatique et ses révolutions, l'objet, Java, CORBA, XML, etc., etc., etc. Mais, lorsque vous intervenez en mission chez des clients, vous oubliez que le métier de ces clients n'a souvent pas vraiment subi de révolutions... tout au plus des évolutions.

Pretons la banque, par exemple. L'exemple de ses systèmes d'informations (avec les comptes client) est frappant. Hier comme aujourd'hui, c'est toujours la même chose. D'un côté, le serveur de données de l'autre l'utilisateur. Et dès les années 70, avec l'apparition de la télématique, l'utilisateur a voulu consulter et mettre à jour ses données à distance, donc en architecture 2iers, pour avoir sa situation de compte à jour!

[Rq : fin 2002, la télématique représente une faible part des activités des SSII, cf. [slide n°4 de la présentation de Desfontaine en SSII](#)]

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

14

## Réflexion

- Aujourd'hui, on a toujours un client léger (appelé Java, par exemple) d'un côté et un serveur de données de l'autre. Le troisième "tier" est le middleware qui distribue les services entre les utilisateurs et gère les transactions ou encore la charge (load-balancing). Le tout est éventuellement agrégréé d'un 4ème "tier". Le serveur de présentation dynamique (ASP, JSP, PHP4...) pour rendre le tout plus joli.

*4 tiers ??? Ça n'existe pas à 4 ou 3 tiers ??*

"Tiers" signifie NIVEAUX ! "Tiers" est un mot anglais!! (synonyme : "level" ou la rigueur "layer"). Tu ne vas pas me dire que tu croyais que "tier" était français, non ???

*Humf... non-non, j'ai rien dit... continuez...*

On est donc passé du VAX/AIX, généraliste l'administrateur, au client-serveur utilisé par l'utilisateur puis à l'intranet/internet qui exporte des services vers le "client" final, via une interface pas toujours très bien pensée... Humf... Et en plus, cette chaque niveau on a introduit un gros plat de spaghetti!

En effet, au début, tout était sur la même machine, maintenant, n couches réseaux viennent se rajouter entre le serveur et le client.

Mais franchement, la problématique de fond n'a pas vraiment changé. Accéder, mettre à jour, traiter et présenter l'information.

*Et qui avait ces besoins dans les années 70 ?*

Mais les mêmes qu'aujourd'hui, mon gars : les banques! Mais aussi toutes les entreprises qui centralisaient (et centralisent toujours) leur comptabilité.

Aujourd'hui, celles qui attirent l'attention sont les Bio-C (Business to Client), mais les possibilités de connexions sont encore trop faibles pour que ce domaine explose complètement. Seul le sa-disant dépassé minitel arrive encore, en France, à tirer son épingle du jeu (avec des coûts de consultation payant, même si on n'achète rien : un coup de génie!) et à rapporter encore des milliards de francs. [Dixit SVM Janvier 2001, le minitel génère 95 millions d'appels par mois et le 3615 (service "Réserve") rapportent plus de 12 milliards de francs, dont le quart (3 milliards) sont réservés aux seuls fournisseurs de services... le jackpot!]

En revanche, le Bio-B, ça fait 30 ans que ça dure...

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

15

## Réflexion

- Dans votre cas, qui étaient les 2 'B' de 'B to B' ?

"Start-up" avait l'heure, éditeur d'un logiciel (programme conçu chez nous et qui tournait chez nous, avec astrinés, garantie de bon fonctionnement, gestion des transactions... ce qui diffère d'un progiciel, conçu chez l'éditeur mais qui tourne chez le client, comme un Word ou un Excel).

N'oublions pas qu'à l'époque, cela avait toute l'informatique qui coûtait cher, (les programmes comme les énormes machines). Dans les années 70, les clients ne pouvaient pas se permettre d'héberger le matériel informatique. Le coût des machines et des infrastructures pour les accueillir et assurer leur fonctionnement était prohibitif.

Dans les années 80, les serveurs se retrouvent au niveau départemental chez les clients (donc dans les grands départements des clients). Aujourd'hui, vu le faible coût de ces machines, celles-ci se retrouvent... partout.

Nous faisons de la gestion de portefeuilles et de contrats, et ce pour 10% des banques de la place financière française, s'il vous plaît.

*Mais, mais mais n'y avait pas encore Internet, à cette époque ?*

PF... Internet! On ne l'a pas attendu pour nos propres lignes spécialisées soit via Transpac, soit via RTC (moyen de connexion toujours très en vogue par des internautes *accueillis* par l'absence de TADSJ, ou de la BLR - Boucle Locale Radio - ou encore des limitations du câble!).

Cela s'appelle de la télématique, gamin. En 79, on enfilait le combiné de téléphone sur un réceptacle adapté de note terminal de saisie, et pour interroger nos comptes on gérait nos transactions en communé... comme le bon vieux minitel!

*A quoi ça ressemblait, l'informatique dans les années 70 ?*

Ha, ça, c'était du brocage de haute voltige : j'ai connu la mécanographie classique, avec l'UNIVAX 32K (évolution de l'...).

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

16

## Réflexion

- D'une part, on n'avait pas de *des* (enfin... ils existaient mais leur usage ne s'est véritablement démocratisé que dans le début des années 80). Avant, tout se faisait avec des machines à écrire dotées de la fameuse sphère d'IBM. C'était les télétypes (TTY) dont les premiers écrans, d'ailleurs, ne faisaient que reproduire le comportement. Encore dans les films de cinéma d'aujourd'hui (il n'est pas rare de voir des messages s'inscrire sur type d'écran avec, en bruit de fond, un son de machine à écrire!!) (cf. [http://www.lesgros.net](#)).

Ensuite, il fallait savoir gérer les aspects fonctionnels d'une application ou avec les aspects techniques du matériel sur lequel elle tournait. En particulier, on devait gérer la mémoire associative, qu'IBM a popularisé sous le nom de mémoire virtuelle (et qui est toujours utilisée par les OS actuels).

Enfin, il fallait alors rentrer une application au clavier. Si on rentrait tout, il n'y avait pas assez de mémoire pour tout compiler, donc on découpait l'application en modules, compilés séparément et qui ne se chargeaient en mémoire que s'il le fallait... bref, l'équivalent des DLL de C++ ou des jar en Java.

Certes, aujourd'hui cesdell ou jar existent évidemment toujours, mais pas pour des problèmes de mémoire (avec 256Mo minimum, ça commence à aller), plus pour des raisons de modularité et d'évolutivité : on remplace un composant sans tout recompiler, ce qui facilite la livraison aujourd'hui... et ce qui nous faisait gagner du temps à l'époque (où une compilation était loin d'être une petite opération banale et peu coûteuse en temps). On n'avait droit qu'à une compilation par semaine sur l'unique machine, et encore uniquement pendant les TP!

*Les TP ? Vous aussi, vous avez des Travaux Pratiques à rendre ?*

Je n'y crois pas !!! Y sai même pas ce qu'est du TP ?? Temps perdu jeune ignorant! Les quelques machines que nous possédions à l'époque comportaient deux cycles d'activités bien distincts.

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

17

## Réflexion

- D'un côté, en journée, le TP où toutes les opérations transactionnelles de type client/serveur s'effectuait. Par exemple, en journée, tous les clients se connectaient et laissaient une demande de relevé de compte qu'ils souhaitaient recevoir par la poste dans les jours suivants. Ensuite, pendant la nuit, les batch prélevaient les relevés. Ces process étaient là pour traiter de très gros volumes d'informations (comme, ici, l'édition de tous les relevés de compte demandés la veille) afin non seulement de les éditer, mais aussi de faire des traitements comme celui de les classer selon l'adresse des clients, afin que la livraison des relevés par la Poste s'effectue de manière plus logique géographiquement... et donc moins coûteuse pour nous.

TP et batch, tels étaient les 2 cycles de nos machines. Et encore aujourd'hui, le DataWorkflow (flux de données) de type "business" est là pour gérer ces process (alors que le DataWorkflow de type coopératif gère la vie des documents au sein d'un groupe, cf. Louis par exemple).

L'UNIVAX était essentiellement mono-tache (en théorie, on pouvait faire du multi-process... à condition de se farcir à la main toute la gestion des interruptions! En pratique, personne ne s'y risquait). Donc, une compilation serieuse, sur les TP. Un test par une. Avec correction directement dans le code machine, donc dans des zones de patch prévues à cet effet!

Cela veut donc dire qu'avant la moindre compilation, nous on ré-*ré*-chi-ssait! Certes, je ne nie pas le fait qu'il soit aujourd'hui bien plus confortable d'écrire du code, d'appuyer sur une touche pour compiler en 2 secondes, puis de tester. Simplement, ce type de comportement poussé à l'extrême fait que j'ai déjà vu des banques écrire des systèmes entiers, packages après packages, sans faire de véritable *...*! Les vrais tests n'intervenant qu'en phase d'intégration... et là en général... tout expose! Voilà ce qui arrive lorsque l'on pousse la politique de "Testai - erreur" trop loin. *[alors que pour de gros projets, elle a parfois du bon, cf. ...]. Ça même article rappelle que la "*

*...* n'est pas ses balbutinements, même en 2002.]

Nos procédés de tests se devaient d'être bien plus rigoureux et le programme d'être maintenable. Pour cela, il fallait bien cerner les interfaces fonctionnelles de chaque composant et veiller aux dépendances.

12/11/2002

COBOL Théorie 2002 / Pierre Renouprez

18

### Réflexion

• **Dll, modularité, évolutivité, dépendance...** que des termes très actuels! C'était quoi, votre problématique client-serveur à vous? Hé bien, on avait des présentations à faire sur des données financières, pour n sociétés différentes, chacune avec un traitement particulière et leur généralisation financière propre. En plus, il fallait pouvoir intégrer rapidement les évolutions de la Loi en matière de finance. Pour les batch, il fallait donc avoir des composants suffisamment simples avec un tronc commun efficace afin de traiter tous les clients en même temps. D'où des EXIT (procédures déclenchées à la fin du traitement d'un composant) personnalisées selon le client, des ressources dynamiques qui se chargent où non, capables de se neutraliser entre elles en cas de problème, avec toutes les procédures de récupération sur erreur. Chaque composant est ainsi capable de traiter un certain volume de données, et c'était cette "volumétrie" qui était facturée au client en fonction du nombre de ses comptes. C'était aussi le cas de l'aspect transactionnel (avec ses points de synchronisation : COMMIT et ROLLBACK) qui était au point même que ces points de synchronisations concernaient le client comme le serveur : ce dernier était, en cas de problème, était capable de faire son rollback et de prévenir le client afin que celui-ci effectue également de son côté son propre rollback.

**Et la programmation ? Ca a quand même changé, la programmation, non ?** Mouti m'intéressait si j'ai connu l'assembleur, n'oublions pas que le C et même le C++ sont 2 langages qui se situent juste au dessus, vous laissant vous dépanner avec les pointeurs et la gestion mémoire. Si l'objet a apporté un regroupement plus clair des données et de leur traitements associés, seuls les langages basés sur un méta-modèle (Smalltalk, Java) apportent un réel plus. Tout le reste n'est que de l'assembleur masqué et rendu plus pratique. Et encore, Java est faiblement typé... c'est le règne du "void" ! Et puis, l'objet, l'objet... De nos jours, le principe de base était la séparation claire du code et données [et cela peut s'illustrer avec ces exemples]. Et ce pour évidente raison de sécurité, et aussi de portabilité. Lorsque l'on migrait d'un système à un autre, la procédure était rodée : on bougeait d'abord la base de données, puis les TP, puis les batch. On est ainsi passé au milieu des années 80 de ADABAS (aqueil il ne manquait que les jointures) à d'autres systèmes sans problème.

### Réflexion

• De toutes façons, en matière de gestion, COBOL (1958 !) reste le roi, même aujourd'hui en 2001. [de même qu'un certains nombres 'd'anciennes techno, comme nous le rappelle en 2002 cet article].

**Oùdà !! COBOL ??? Ca existe encore ?** Comment ça, « ça existe encore » ??? Mais qu'est-ce que tu crois, jeune galopet ? Qui on peut se permettre, dans la finance, de dire que 1-1 font 1 999974878978 à cause d'un processeur récalcitrant ? La notation flottante scientifique n'est pas faite pour la gestion. Un seul centime de décalage au début d'une opération et ce sont des millions de francs envoyés à la fin.

COBOL, avec son système de virgule virtuelle, ne travaille que avec des entiers qui viennent se mettre dans des champs prédéfinis, dont on a précisé le nombre de chiffres de la partie entière et de la partie décimale. Comme ça, on est tranquille! Enfin presque, parce que IBM et Intel n'ont pas été fous de se mettre d'accord sur la représentation des nombres entiers!!! Lun le code "poids fort, poids faible", l'autre "poids faible - poids fort".

**Et ça passe l'an 2000, ça ???** Mrrmf... Mort de rire!!! Mais l'an 2000, jeune sot, cela fait plus de 30 ans que l'on s'en occupe!!! Par contre, cristin, qu'il ne s'est finalement rien passé le 1er janvier 2000, hein ? [cf. 2000 - bug or not ?].

Comment pensiez-vous que l'on codait les dates dans des applications financières dans les années 70 alors que l'on traitait de contrat de prêt sur 30 ans ?... (s8 , 1975>30 > 2000). Toute date était codée en jour. Je crois qu'on devrait avoir un problème vers 2176, pas avant!

De toutes façons, le calcul est vite fait : la référence était le 1er janvier 1901 (pour ne pas avoir à s'occuper de l'année non-bissextile 1900, puisque les années bissextiles sont celles divisibles par 4 sauf celles divisibles par 100 sauf celles divisibles par 400, donc 1600, 2000 sont bissextiles, pas 1900 ou 2100). Le nombre de jours maxi que l'on stocke est 99999. Vous divisez par 365,25 jours et vous obtenez... 273 ans 9 mois et 11 jours. Bon, allez, 2174, j'étais pas loin.

### Réflexion

• **Mais c'était quoi, alors, toute cette histoire autour de l'an 2000 ???** Haphorhop... pas de conclusion hâtive, s'il vous plaît... D'abord, je n'ai jamais dit que nous les systèmes avions anticipé cela et codé les dates comme nous.

Ensuite, le vrai chantier qui se cachait derrière ce bug, c'était celui d'avoir une véritable plate-forme de test, qui reproduise les conditions de la "prod" (production), avec tous ses flux financiers. YZK n'était qu'une formidable opportunité de mettre en place un véritable environnement de test qui allait servir, certes, à tester le passage à l'an 2000, mais qui sert également de véritable plateforme d'intégration avant mise en production. Et la mise en place de tel environnement, cela coûte très cher, même si par ailleurs pas une ligne de code ne doit être réécrite pour corriger le problème du passage à l'an 2000. [cf. aussi article].

**Pour reprendre votre propos initial, l'informatique, telle qu'il l'a utilisée par le métier (ici bancaire) n'a pas changé... vous pouvez détailler ?** Et non ça n'a pas changé... Tous les traitements de base de mise à jour et d'accès aux systèmes d'informations bancaires sont toujours inchangés depuis 30 ans, en général en COBOL (bon, COBOL 2, d'accord) [cf. la blague 2000 - 10000].

Réfléchissez un peu sur le nombre de ligne de code qui, en 30 ans, a vraiment été réécrit. Ce n'est pas évident de vraiment tout repenser, car il faut déjà à arriver à connaître l'existant. Les Systèmes d'Informations complexes possèdent des fonctions conçues pour un fonctionnement entre 5 et 7 ans. C'est normal : dans l'industrie, on principe, on renouvelle son matériel en gros tous les 5 ans, après l'avoir aimé! Hé bien là, chez les grands clients, comme les banques, non! La plupart de ces vieilles fonctions tournent toujours aujourd'hui.

Où, le problème, c'est que le turn-over des programmeurs est en gros de 20% tous les 2 ans. La documentation devient vite périmée au fur et à mesure des modifications induites ne serait-ce que par les changements de législation financière.

Certes, il existe dans ces grands groupes des schémas directeurs qui permettent de s'adapter à des objectifs variables et qui évoluent dans le temps... mais l'investissement est souvent considérable, le plus simple étant alors de différer les changements requis!

### Réflexion

• Donc non seulement la problématique ne change pas, mais les moyes ns tant matériels que fonctionnels n'ont que peu évolué! Les applications récentes se contentent souvent de hériter en encapsulant de l'ancien (le legacy (que vous retrouverez présente dans l'article sur les applications)). Ainsi, si l'on creuse derrière vos jolis J2EE sessions et vos points (Java Enterprise) ou autres systèmes de distribution de services et d'applications... si l'on va derrière tout ça, il n'est pas rare de trouver des accesseurs COBOL [d'où de gros problèmes... aussi pour les applications].

Cela change d'autant moins que l'on est obligé de garantir une compatibilité ascendante à tous les degrés... et si la base "marche" (même basée sur des techno anciennes, mais fiables, robustes et éprouvées), hé bien on ne va pas la changer juste pour le fun. Le reste de la pyramide des applications "modernes" est basé dessus!

**Mais... cela ne vous tente pas de mettre à jour tout ça : les techniques informatiques ont quand même évoluées, non ? A moyen terme, il y gagnerait (au moins en maintenance), non ?** Ha... l'insouciance de la jeunesse... la passion, tout ça... Sauf que, petit, tu oublies une chose : Le client pour lequel tu intervient, son métier, c'est pas l'informatique, c'est la banque. Et son métier, dont l'origine remonte à bien avant l'informatique, ça pas fondamentalement changé. Simplement, sa société s'est dotée de différents départements informatiques, c'est tout.

[...] rappelle d'ailleurs qu'il n'y a pas de banque intrinsèque dans le changement : on ne peut pas changer juste pour le plaisir de changer.]

Je te l'accorde, le problème de ces grands clients (banque, assurance, centre de comptabilité, etc.), c'est qu'ils n'ont pas de vision technique à moyen ou long terme. Toujours dans le domaine technique, s'ils ont des schémas directeur, ils les ignorent royalement.

Pourquoi ? Parce que pour ces sociétés, cela ne sert à rien d'investir pour être le meilleur (techniquement) dans 5 ans. Ils veulent juste être maintenant.

Du coup, l'informatique n'est tolérée que si : - elle rapporte tout de suite de l'argent, ou - elle permet de faire des économies (comme le calcul de la tournée postale des relevés de compte).

### Réflexion

• En plus, comme chaque département informatique gère son propre budget et présente son propre bilan, il n'est pas rare de les voir accepter un projet qui ne leur coûte pas trop cher, même si cela va coûter le double aux départements voisins qui devront s'adapter au nouveau système!

Et quand bien même une solution informatique est retenue, sachez bien petit que ce n'est pas forcément la meilleure ou la moins chère. C'est avant tout celle qui est la plus "irréprochable", c'est-à-dire fournie par le "n°1" dans son domaine. Même si la solution se révèle mauvaise, personne ne veut reprocher d'avoir choisi un leader. Alors que si vous choisissez une solution plus risquée, plus moderne... et qui se plante, vous saluez avec.

Et ça, cristin, on ne le voit pas que en informatique. L'exemple le plus célèbre reste le choix du standard VHS pour les cassettes vidéo...

Reste la question... combien de temps un tel système (basé sur un socle vieux de 30 ans et empillé des couches technologiques plus récentes) va tenir ? Qui viendra mettre un coup de pied dans cette fourmilière ?

**Toutes ces technologies actuelles, modulaires, dynamiques pour faire finalement la même chose qu'avant mais de façon distribuée... Et en plus pour masquer un existant monolithique de 30 ans !... J'ai un petit coup d' barre moi, tout d'un coup.**

Allez petit, rejoins-tu! Songe à toute la complexité que ces couches successives apportent. Complexité qu'Internet a décapulée, avec ses nombreux protocoles et langages (JavaScript, différent selon les browsers et encore différent des applets java, elles-même n'ayant rien à voir avec VBScript, le tout utilisant un HTML qui n'est rien d'autre que du SGML mal formé alors que XML est bien formé, le tout présenté de façon dynamique par des servletset JSP, ou encore par de l'ASP ou du PHP4 ou... etc., etc.).

Bref, vous n'êtes pas prêt de manquer de travail : vous travaillez dans un des rares métiers où on ne peut pas se laisser abattre, tu dois Comprendre toutes ces couches successives et complexes, tu devras, Facturer très cher tes journées d'intervention de consultant, tu pourras! [et cela évoque un avis curieusement similaire de ce forum en 2002].

Il fait bien remarquer que "celui" (- toujours plus de complexité -) est à la base d'un certain mécontentement général dans ce métier !