COBOL cours 9

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

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.

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.

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 du 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

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.

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 séquencement 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.

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 incrémentations et décrémentations. DIVIDE est autorisé pour calculer un reste.
- PR-11 : Les parenthèses doivent être utilisées pour préciser l'ordre d'évaluation d'une condition complexe.

Programme COBOL

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

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.

Exercice 1 1/2

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

Exercice 1 2/2

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

Exercice 2

- Supposons que l'on veuille décrire 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 définie 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

Réflexion:

http://dchaffiol.free.fr/info/developpeur/art_toujoursPareil_t.htm

« Mais si... 30 ans qu'on fait la même chose que j'vous dis ! »(1970-2000)

- **Vous êtes jeune, vous êtes "in", branché, dans l'coup, contemplant 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 vexe.
  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 regardiez 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 soulignés
- *30 ans de métier... et vous êtes diplômé en informatique ??? Dans les années (19)70 ?*
  Et comment, je l'suis, petit. Deuxième promotion d'un des trois premières écoles d'ingénieur en informatique de Paris, s'il vous plait. 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.

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 Gaule (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) était créée par une Loi de novembre 1967. Le Pascal venait d'apparaître en 1969 et en 1970... le parc 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 voyons. Vous, les jeunes, vous voyez toujours l'informatique et ses révolutions, l'objet, Java, CORBA, XML, etc., etc... Mais, lorsque vous intervenez en mission chez des clients, vous oubliez que le métier de ces dits clients n'a souvent pas vraiment subi de révolutions... tout au plus des évolutions.
  Prenons 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 2tiers, pour avoir sa situation de compte à jour!
  [Rq : fin 2002, la télématique représente une faible part des activités des SSII, cf. slide n° 6 de la présentation "Pyramides en SSII" *]*

Réflexion

- Aujourd'hui, on a toujours un client léger (applet 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 agrémenté d'un 4ème "tier", le serveur de présentation dynamique (ASP, JSP, PHP4,...) pour rendre le tout plus joli.
  4 tiers ??? Ça n'existe pas 4 fois "1 tiers" ?!
  "tiers" signifie NIVEAUX ! "Tiers" est un mot anglais!! (synonyme : "level" ou à la rigueur "layer"). Tu ne vas pas me dire que tu croyais que "tier" était français, non ???
  Hmpf... non-non, j'ai rien dis... continuez...
  On est donc passé du VAX AIX, géré par l'administrateur, au client-serveur utilisé par l'utilisateur puis à l'intranet-internet, qui exporte le service vers le "client" final, via une interface pas toujours très bien pensée... Humpf.. Et en plus, entre 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 même 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 Bto-C (Business to Client), mais les possibilités de connexions sont encore trop faible pour que ce domaine explose complètement. Seul le soi-disant dépassé minitel arrive encore, en France, a tiré 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'appel par mois et le 3615 (service "Kiosque") rapportent plus de 12 milliards de francs, dont le quart (3 milliards) sont réservés aux seuls fournisseurs de services... le jackpot!]***
  En revanche, le Bto-B, ça fait 30 ans que ça dure...

Réflexion

- *Dans votre cas, qui étaient les 2 'B' de 'B-to-B' ?*
  D'un côté, des banques, de l'autre nous, une "start-up" avant l'heure, éditeur d'un logiciel (programme conçu chez nous et qui tournait chez nous, avec astreintes, 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, *c'était 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* de ces clients).
  Aujourd'hui, vu le faible coût de ces machines, celles-ci se retrouvent... partout.
  Nous faisions de la gestion de portefeuilles et de contrats, et ce pour 10% des banques de la place financière française, s'il vous plaît.

  Mais, maismaisil n'y avait pas encore Internet, à cette époque ?
  Pff... Internet! On ne l'a pas attendu pour tirer nos propres lignes spécialisées soit via Transpac, soit via RTC (moyen de connexion toujours très en vogue par des internautes *actuels* frustrés par l'absence de l'ADSL ou de la BLR - Boucle Locale Radio - ou encore des limitations du câble!).
  Cela s'appelle de la télématique, gamin. En 79, on enfichait le combiné de téléphone sur un récepteur adapté de notre terminal de saisie, et pour interroger nos comptes on gérait nos transactions en commuté... comme le bon vieux minitel!

  A quoi ça ressemblait, l'informatique dans les années 70 ?
  Ha ça, c'était du bricolage de haute voltige : j'ai connu la mécanographie classique, avec l'UNIVAX 32K (évolution de l'UNIVAC).

Réflexion

- D'une part, on n'avait pas d'écran (enfin... ils existaient mais leur usage ne s'est véritablement démocratisé que dans le début des années 80). Avant, tout se fai sait 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 ce type d'écran avec, en bruit de fond, un son de machine à écrire!!! (cf. L'informatique au cinéma)
  Ensuite, il fallait savoir gérer les aspects fonctionnels d'une applicati on 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 chausse-pieds! Si on rentrait tout, il n'y avait pas assez de mémoire pour tout compiler, donc on découpait l'applica tion en modules, compilés séparément et qui ne se chargeaient en mémoire que s'il le fallait... bref, l'équivalent des dll du C++ ou des jar en java.
  Certes, aujourd'hui ces dll 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 aviez des Travaux Pratiques à rendre ?
  Je n'y crois pas !!! Y sait même pas ce qu'est du TP !? Temps Partagé jeune ignorant! Les quelques machines que nous possédions à l'époque comptaient deux cycles d'activités bien distincts.

Réflexion

- D'un côté, en journée, le TP où toutes les opérations transactionnelles de type client/serveur s'effectuaient. 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 prenaient le relais. Ces process étaient là pour traiter de très gros volumes d'informations (comme, ici, l'édition de l'ensemble des 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. Lotus 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 compil par semaine, sur les TP. Et un test par jour. 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-flé-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 jeunes écrire des systèmes entiers, packages après packages, sans faire de véritable testunitaire! Les vrais tests n'intervenant qu'en phase d'intégration... et là en général... tout explose! Voilà ce qui arrive lorsque l'on pousse la politique de "l'essai - erreur" trop loin. *[alors que pour de gros projet, elle a parfois du bon, cf. Linux ! Ce même article rappelle que la "*science informatique*" n'en est qu'à ses balbutiement, 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.

Réflexion

- *Dll, modularité, évolutivité, dépendance... que des termes très actuels! C'était quoi, votre problématique client-serveur à vous ?*

Hé bien, on avait des traitements à faire sur des données financières, pour n sociétés différentes, chacune avec leur présentation particulière et leur réglementation 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 souples 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.

Concernant le TP, l'aspect transactionnel (avec ses points de synchronisation : COMMIT et ROLLBACK) était géré 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 ?

Moui, m'enfin 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 dépêtrer avec les pointeurs et la gestion mémoire.

Si l'objet a apporté une 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 mon temps, le principe de base était la séparation claire du code et données! ***[et cela peut s'illustrer avec ces kangourous mutants!]*** Et ce pour d'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 donnée, puis les TP, puis les batch. On est ainsi passé au milieu des années 80 de ADABAS (auquel 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 technos, comme nous le rappelle en 2002 cet ingénieur.]*

Qwâûâ !? COBOL ??? Ca existe encore ?

Comment ça, « ça existe encore » ?? Mais qu'est-ce que tu crois, jeune galopiot ? Qu'on peut se permettre, dans la finance, de dire que 1+1 font 1.9999748789378 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 envolés 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 chiffre 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 foutu de se mettre d'accord sur la représentation des nombres entiers!!! L'un le code "poids fort, poids faible", l'autre "poids faible - poids fort".

Et ça passe l'an 2000, ça ???

Mrrmff.. Mort de rire!!! Mais l'an 2000, jeune sot, cela fait plus de 30 ans que l'on s'en occupe!!! Pourquoi croisnu qu'il ne s'est finalement rien passé le 1er janvier 2000, hein ? ***[cf.*** Y2K : bug or no bug ***]***

Comment penses-tu 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 ?... (ssi , 1975+30 > 2000). Toute date était codée en jour. Je crois qu'on devrait avoir un problème vers 2176, pasavant!

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 autours de l'an 2000 ???*

Hophophop... pas de conclusion hâtive, s'il vous plaît...

D'abord, je n'ai jamais dis que *tous* les systèmes avaient anticipé cela et codé les dates comme nous.

Ensuite, le vrai chantier qui se cachait derrière ce bug, c'était celui de monter une véritable plate-forme de test, qui reproduise les conditions de la "prod" (production), avec tous ses flux financier. Y2K 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 plate-forme d'intégration avant mise en production. Et la mise en place d'un 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*** 2K22K1 ***]***

Pour reprendre votre propos initial, l'informatique, telle qu'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*** Y2K : COBOL ***]***

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 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, en principe, on renouvelle son matériel en gros tous les 5 ans, après l'avoir amorti. Hé bien là, chez les grands clients comme les banques... non! La plupart de ces vieilles fonctions tourne toujours aujourd'hui.

Or, 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 moyens tant matériels que fonctionnels n'ont que peu évolué! Les applications récentes se contentent souvent de bâtir en encapsulant de l'ancien (le ***legacy*** *[que vous retrouvez présenté dans l'article* Bossaci l'Architecte *]*). Ainsi, si l'on creuse derrière vos jolis J2EE sessions et components (Java toEnterprise) ou autres systèmes de distribution de services et d'applications... si l'on va derrière tout ça, il n'est pas rare de trouver desaccesseurs COBOL! *[d'où de gros problème... aussi pour les architectes]*

Cela change d'autant moins que l'on est obligé de garantir une compatibilité ascendante à tous les étages... et si la base "marche" (même basée sur des technos 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 basée 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 interviens, son métier, c'est pas l'informatique, c'est la banque. Et son métier, dont l'origine remonte à bien avant l'informatique, n'a pas fondamentalement changé. Simplement, sa société s'est dotée de différents départements informatique, c'est tout.

[Linus Torvalds rappelle d'ailleurs qu'il n'y a pas de bénéfique 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 meilleurs (*techniquement*) dans 5 ans. Ils veulent juste l'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 d'échelle (comme le calcul de la tournée postale des relevés de compte lorsque plus rien).

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, sache 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 vous reprochera d'avoir choisi un leader. Alors que si vous choisissez une solution plus risquée, plus moderne,... et qui se plante, vous sautez avec.

Et ça, croismoi, 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 empilant 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, réjouis-toi! Songe à toute la complexité que ces couches successives apportent.

Complexité qu'Internet a décuplée, 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 au monde qui contribue à sa propre continuation en se créant toujours plus de problèmes!

Allons jeune Jedi, ne pas se laisser abattre, tu dois! Comprendre toutes ces nouvelles technologies complexes, tu devras. Facturer très cher tes journées d'intervention de consultant, tu pourras! [et cela évoque un avis curieusement similaire de ce jeune ingénieur en 2002] [Laurent Bossaci Coach XP fait bien remarquer que "cela" (- toujours plus de complexité -) est à la base d'un certain mécontentement général dans ce métier !]