SISTEMAS DE COMPUTACIÓN I

|  |  |
| --- | --- |
| Módulo I  Unidad I | Comprender los principios básicos del procesamiento de datos |
| Procesos de datos: historia y automatización |
|  | Trabajo práctico Requerido Nº 1.  **Una aproximación histórica: proyectemos el futuro conociendo sus orígenes** |

# Presentación

Este trabajo tiene como propósito permitir la reconstrucción histórica del algún tema clave del campo de la computación para que, a través de su análisis, logre elaborar conclusiones que le permitan construir una mirada prospectiva[[1]](#footnote-1) sobre él.

Para poder alcanzar este objetivo usted deberá haber leído la bibliografía propuesta en el Orientador del aprendizaje correspondiente a la Unidad I.

# Consignas

1. Seleccione un tema del campo de la computación que sea de su interés. (vinculación entre tecnologías afines, potencia de procesamiento, empleo de distintas técnicas de tratamiento, avances en el desarrollo del Software, etc.)
2. Confeccione una "línea de tiempo" de acuerdo con el eje temático de su elección.
3. Determine las innovaciones y continuidades a lo largo de la historia.
4. Explicite de una manera creativa conclusiones, tendencias y explicaciones acerca del futuro inmediato en el campo.
5. Incluya un glosario de los términos que considere necesarios.

# Criterios de corrección

En la corrección de este Trabajo Práctico, tendremos en cuenta los siguientes criterios:

* 1. Pertinencia de los conceptos utilizados.
  2. Claridad y coherencia en la argumentación escrita.
  3. Entrega en tiempo y forma.
  4. Originalidad

*Utilice estos criterios para anticiparse a los resultados de la evaluación. Adecue su producción a los parámetros señalados. Si tiene dudas, consulte a su tutor/a!*

Buenas noches.

La verdad, cuando vi el trabajo que había que hacer, me quedé sorprendido pues sé de antemano que relatar una historia con una mirada anticipatoria (prospectiva) no es poca cosa ni sencilla ni mucho menos escueta. Mi temor radica en que, aunque intente disciplinadamente ser lo más breve posible, no consiga algo que sea considerado así, breve. Pero lo prefiero a caer en la falacia lógica del reduccionismo, de la sobre simplificación.

He visto comentarios de compañeros que hablan de "graficar" una línea de tiempo. En mi humilde entender, un gráfico para un trabajo tan interesante y apasionante y complejo como este no serviría más que para un mero índice. Puedo estar equivocado. Después de todo, es tan solo mi punto de vista y estoy aquí para aprender, así que no doy por sentado que tengo razón.

Mi interés, obviamente por la elección de esta carrera universitaria, tiene que ver con la ingeniería de software (que tiene que ver con la historia, pero particularmente, con el por qué se adoptaron ciertas soluciones y otras no), y en especial, con la arquitectura de software (la noción de que es posible hacer más eficiente algo que posiblemente ya sea eficaz, por ejemplo). Y obviamente, la literatura que trate sobre el tema siempre me ha interesado.

Al ver la propuesta de este práctico, me vino inmediatamente a la mente uno de los libritos que más me ha impresionado (y que entiendo versa sobre este mismo tema): Memorias de un Viejo Informático. Es un libro de un informático español que inicia su carrera en los '70, y que coincidentemente, es una década que abarca la película recomendada por la cátedra, Pirates of Silicon Valley. Debo aclarar que, en mi concepto personal, en esa década se sentaron las bases de lo que hoy conocemos como la informática de masas, de la PC en la que escribo estas líneas. Y la historia de este informático resulta de lo más interesante por haber estado toda su carrera "en las grandes ligas". Si hay una forma de saber de qué trata realmente la informática, pues nada, con un Steve Jobs no alcanza. La informática, si se puede decir que fue determinante, es porque era la forma de concretar proyectos enormes. Pero bueno, esto se verá en el texto de este trabajo.

El presente trabajo recupera mucho de lo escrito en ese librillo, pero interpolado con mis propias observaciones, y el resumen en sí es de mi autoría pues he debido seleccionar lo que me ha parecido pertinente, y sinceramente espero que, a pesar de lo extenso, pueda verse que he puesto mi mejor esfuerzo en explicar, tal vez usando las palabras de otro, lo que los informáticos hemos venido haciendo, y las condiciones que condicionan las tecnologías.

Al respecto, no dejaré la cosa en suspenso: no veo en las páginas de la historia que la tecnología se decante siempre por la mejor propuesta. No. Las elecciones están condicionadas por muchos factores. No es solo el usuario. Son los tiempos, la economía, las estrategias marketineras, las modas, y un largo etcétera. Pienso que es muy ingenua la percepción de plantear que lo que hoy vemos en las computadoras es una evolución natural de las mismas. La informática (en muchas ocasiones) ha tenido que avanzar al compás de una marcha forzada, a veces desordenada, incluso tuvo que adoptar esquemas que su propia conciencia desaconsejaba.

Pero no nos adelantemos. Escuchemos el relato y lleguemos a nuestras propias conclusiones.

# 1970

En Sexto de Bachillerato (Plan 1957), o sea, con dieciséis años, y a principios de los setenta, había que decidir varias cosas: Una: si iba a estudiar una Carrera Universitaria o me iba a poner a trabajar. Decidí entonces, no sé muy bien por qué, que quería ser "Especialista en Cerebros Electrónicos", más o menos. Los "Cerebros Electrónicos" comenzaban a estar de moda, gracias a las películas americanas en las que se podían ver aquellos sofisticados engendros dominando el mundo... cuando lo que aparecía en pantalla era una unidad de cinta magnética llena de lucecitas, con la cinta girando p'atrás y p'alante todo el tiempo, como si la cabeza lectora tuviera problemas y no fuera capaz de leer correctamente la cinta (lo curioso es que esta práctica de poner una cinta magnética girando estúpidamente a un lado y a otro como imagen de superpotentes ordenadores persistió durante muchos años en las pelis de acción). Además, todo el personal iba con bata blanca, así que debía ser una ocupación realmente importante.

# 1972

Me presenté al examen de ingreso, sorprendentemente aprobé, y allí me planté en octubre de 1972 a comenzar los estudios de Informática... fuera eso lo que fuese. Por cierto, el examen de ingreso era un lujo de modernidad: tipo test, marcabas a lápiz, para cada respuesta, una de las cuatro opciones disponibles en su casilla correspondiente, y luego la corrección la hacía ¡¡Un Ordenador!! leyendo las marcas ópticas de los exámenes: eso, en 1972, parecía extraído directamente de la NASA.

En aquellos tiempos, no había titulados oficiales en Informática, ni siquiera fuera de España, así que los únicos profesores disponibles eran profesionales que tuvieran experiencia en los temas que impartían, y que les gustara enseñar (y supongo que tener algún ingreso extra). Casi todos los profesores eran responsables de Departamentos en las principales compañías informáticas de la época: IBM, Univac, Bull, NCR..., o bien responsables del Departamento de Proceso de Datos de grandes empresas o instituciones que adoptaron tempranamente el uso de ordenadores en España: Ministerios, RENFE, Iberia, Telefónica. Un ejemplo: en Segundo de Carrera teníamos una asignatura de nombre "Informática Básica II". El profesor que me tocó en suerte era Técnico de Sistemas en Univac (Univac se había fusionado con Sperry-Rand Corporation, y se llamaba entonces Sperry-Univac; años después se fusionó con Borroughs, para dar origen a Unisys, compañía que sigue funcionando en la actualidad). El primer día de clase, nos dijo: "Miren Vds., no sé muy bien qué tengo que contarles en esta asignatura, pero yo de lo que entiendo es del Univac 1110, así que les voy a contar cómo es y cómo funciona un Univac 1110. Y para que sepan Vds. cómo funcionan otros ordenadores, a mediados de curso vendrá un colega mío que les contará cómo es y cómo funciona el IBM 370...". Y eso es lo que hizo. El examen final consistió en escribir en ensamblador un programa de canal para el Univac 1100 que fuera capaz de leer un disco magnético con acceso directo.

Seguro que algunos os estáis preguntando: ¿Cuál sería el currículum de semejante pseudo-Carrera? Pues en Primero de Carrera eran cuatro asignaturas:

* Informática Básica I (donde te decían qué era un ordenador, para qué servía, qué era un disco, una cinta... nadie tenía ni idea de ordenadores al sentarse el primer día en clase).
* Métodos Matemáticos I (en realidad era Algebra Lineal, pero bastante morigerada sobre lo que se daba en Ingenierías).
* Inglés I (que empezaba por el "I am, you are,..." En España en aquellos tiempos el "idioma Moderno" que se estudiaba en Colegios e Institutos era francés, comment allez vous?).
* Programación I, subdividida en tres sub-asignaturas a su vez: Fortran, Cobol y Ensamblador (del UNIVAC 9200, IBM-like).

Como veréis, no se perdía el tiempo con tonterías, todo lo que se daba al principio era directamente utilizable para ponerte a trabajar al día siguiente de acabar Primero... y muchos lo hacían.

En Segundo, había Informática Básica II, Métodos Matemáticos II (Cálculo, igual de morigerado que en Primero), Inglés II (ahora ya tenías que saber al menos el to be), Programación II (Ensamblador más avanzado y Algol, un lenguaje precursor de Pascal y con menos éxito todavía), y Metodología (en realidad era Lógica).

Quizá pudiera parecer que los estudios en aquel Instituto de Informática eran como los de una Súper-Academia de las que tanto proliferaban en la época... pues nada más lejos de la realidad. Es cierto que Primero y Segundo Curso eran eminentemente prácticos, siendo Informática Básica y los diversos lenguajes de Programación las asignaturas que más tiempo consumían (junto con Álgebra, en primero, y Cálculo, en segundo).

Pero a partir de Tercero, la orientación cambiaba: Estadística, Análisis Numérico y Teoría de la Información y la Codificación eran las asignaturas clave de Tercero (en ésta última ya se estudiaban los códigos de Huffman, por ejemplo), junto con Teleproceso, Algoritmos (Tecnología de la Programación era el nombre de la asignatura) e Inteligencia Artificial, que tampoco es que hubiera mucho en esta área en la época, pero se estudiaba lo que había.

En Cuarto, el coco era una asignatura preciosa (para mi gusto degenerado) que se llamaba Técnicas de Optimización de Sistemas, y que tenía dos partes: Teoría de Colas (sucesos estocásticos, distribuciones de Poisson, etc.) y Programación Lineal y Dinámica (el Simplex o método de Dantzig, sobre todo). Además, había más Teleproceso, más Inteligencia Artificial (Autómatas, Gramáticas y Lenguajes), etc. Y en Quinto, más Teleproceso, Sistemas de Recuperación de Información (nombre esotérico de lo que ahora se llamaría "Bases de Datos"), Diseño de Compiladores, Reconocimiento de Formas, y Teoría de Juegos, sobre todo, del Ajedrez, aunque el estudio de un juego tan complicado estaba literalmente en pañales en 1977; había la creencia general de que nunca jamás de los jamases un programa de ordenador podría llegar a vencer a un Gran Maestro... menos de veinte años después, Deep Blue venció no a un Gran Maestro, sino al flamante Campeón Mundial, uno de los mejores jugadores de la Historia, y en el culmen de su juego: Gary Kasparov.

Sin embargo, los medios materiales eran patéticamente escasos. Usábamos el ordenador del Ministerio de Educación, un Univac 9200, que era un casi clónico del IBM 360. Las prácticas consistían en escribir un programa de cada lenguaje (Fortran, Cobol, Assembler) que se llevaban a perforar (sin verificación, claro), compilar y si, sorprendentemente, no tenía errores de compilación, ejecutar... que nunca se llegaba a ese paso, claro. Estoy refiriéndome a UN programa. Al año. No había capacidad para más.

# 1975

En 1975 comencé a trabajar de programador en un gran Banco. El equipamiento del Banco era realmente impresionante: Dos ordenadores NCR Century 200, con nada menos que 32 Kbs de memoria cada uno, dos discos cada uno, tres unidades de cinta magnética (dos de 1600 bpi y la otra de 800 bpi, que se utilizaba como entrada de datos), lector de tarjetas perforada y una impresora realmente rápida cada ordenador. Además, la instalación tenía un switch que permitía asignar discos o cintas físicamente conectados de un ordenador al otro, por lo que en caso de necesidad un ordenador podía correr un programa con seis cintas magnéticas y cuatro discos simultáneamente. Que yo sepa, nunca tuvimos tal necesidad, pero, en fin... No esperéis ver muchas fotos o diagramas de estos viejos ordenadores: Es dificilísimo encontrar nada interesante de ellos en la red, es como si incluso la propia NCR se hubiera olvidado de ellos.

**La CPU.** El NCR Century 200 era una evolución del Century 100, con mayor capacidad de proceso, y mayor capacidad de memoria. No sé si aparecía en algún sitio de la documentación técnica cuántos Megahercios tenía, que en realidad sería algún ciento de Kilohercios, ni cuantas instrucciones por segundo era capaz de ejecutar, que no serían más allá de algún centenar de miles. Por ejemplo, las instrucciones de multiplicar y dividir (aunque tenían su propia instrucción ensamblador) se ejecutaban por software, ni siquiera por firmware, por un procedimiento muy análogo al que usamos los humanos para multiplicar o dividir (los que se acuerden, claro). Su ensamblador era bastante potente, de hecho, más que el de su principal competencia de la época, el IBM 360.

**Lenguaje de Programación.** Su lenguaje nativo era NEAT/3 específico de NCR: Una curiosa mezcla entre Ensamblador y Cobol, con los inconvenientes de ambos y pocas de sus ventajas (es mi opinión: sin embargo, había fervientes defensores del NEAT/3 como el mejor lenguaje inventado jamás: como siempre, para gustos hay colores). En realidad, era una evolución del ensamblador, y, santo y seña de NCR, durante mucho tiempo fue el único lenguaje que admitía esta máquina. Lo mejor que tenía el NEAT/3 eran sus potentes instrucciones para usar tablas internas, que no tenían parangón en otros lenguajes. Sin embargo, a principios de los setenta, NCR implementó por fin Cobol para la gama Century, y yo, de hecho, apenas programé nada en NEAT/3, comenzando casi desde el principio en Cobol. En este lenguaje (Cobol), pese a estar hoy en día denostado y que no lo enseña casi nadie, están programados la mayoría de Sistemas de Información que gobiernan el mundo.

**Consola del Sistema.** Un panel con diales y lucecitas que se encendían y apagaban en función de la dirección en que se ejecutaban las instrucciones, su contenido, y diversas luces para comunicar errores y cosas así. A diferencia de las consolas de lucecitas actuales, que sólo tienen utilidad (y poca) para los técnicos de mantenimiento, la consola se operaba habitualmente para diversas funciones, y había que ser capaz de interpretar lo que el sistema comunicaba con sus luces. Así si las luces de contenido estaban en la situación: Apagada-Apagada-Encendida-Encendida-Apagada-Apagada-Apagada-Encendida, esto quería decir "00110001", es decir, "31" en Hexadecimal, o sea, "1" en ASCII, o "49" en binario, etc.

**Memoria.** Como dije, 32 Kbs (sí, 32.768 bytes, ni uno más). Eran de ferrita. Las ferritas estaban en un armario enorme todito lleno de cables. Era con mucho el elemento más caro del sistema (las ferritas se enhebraban una a una, a mano, y en 32 Kbs había 262.144 ferritas, que son muchas). Al año más o menos de estar yo trabajando, el Banco se gastó una fortuna (creo recordar que unos veinte millones de pesetas de la época en cada ordenador), para ampliar la capacidad de sus dos ordenadores de 32 a 64 Kbs. Pero es que con este cambio, se pudo actualizar el Sistema Operativo también, pasando del B1 original que funcionaba con las 32 Kbs, al B3. ¡Y el B3 permitía multiprogramación! Es decir, con 32 Kbs el ordenador sólo podía ejecutar un programa al tiempo, mientras que con 64, se podían hacer dos particiones de 32 Kbs y correr dos programas simultáneamente, reduciendo casi a la mitad el tiempo total necesario para ejecutar los diversos procesos. Tengo que decir aquí que el Sistema Operativo ocupaba en memoria menos de 5 Kbs, con lo que tenías nada menos que 27 Kbs y pico para tu programa... ¡Una barbaridad de espacio!

**Discos Magnéticos.** Eran removibles, es decir, en cada unidad de disco se ponía uno u otro disco según la necesidad. Estos discos removibles eran cilíndricos, del tamaño de una tartera grande (unos treinta y cinco centímetros de diámetro y unos quince de alto), y la increíble capacidad de algo más de 4 Mbs. (Sí, no me he equivocado, he dicho 4 Mbs, o sea, cuatro millones de caracteres. Y pico, eso sí). La unidad de disco era como una lavadora (de hecho, más grande que una lavadora), con una tapa superior que se levantaba y permitía introducir o extraer el disco mediante un ingenioso dispositivo adaptado en la funda, de tal forma que el delicado contenido del disco rara vez sufría daños. Una vez introducido el disco en la lavadora (todos la llamábamos así), le dabas al botón de "Start" y esperabas sus buenos tres o cuatro minutos a que alcanzara la velocidad de rotación requerida y se pusiera "Ready" (encendiéndose el inevitable pilotito verde), momento en el que el disco estaba operativo. Para extraer el disco, dabas al botón de "Stop" (rojo, como no podía ser de otro modo) y esperabas sus otros dos o tres minutos a que se parara para poder abrir la unidad y extraerlo. Podéis imaginar que, con semejante capacidad, los discos se reservaban para el software (los programas propiamente dichos), y algunos ficheros de datos fijos (Sucursales, Condiciones, Fechas y así). Los ficheros Maestros (Clientes, Cuentas, Empleados, etc.) y los de Movimientos eran bastante más grandes que eso, así que no cabían en los discos. Entonces, seguro que os estaréis preguntando: ¿Pero... dónde se guardaban los ficheros importantes (bueno, y los menos importantes también)? Pues en cinta magnética.

**Cintas Magnéticas.** Armarios enormes con dos carretes, uno a la izquierda y otro a la derecha (había modelos de otros fabricantes que tenían los carretes arriba y abajo) y un sofisticado mecanismo de lectura. El operador colocaba en el carrete de la izquierda la bobina de cinta magnética con el fichero a leer o escribir, extraía (desenrollando la bobina) el comienzo de la cinta y, pasándola a través del mecanismo de lectura (que funcionaba a base de hacer un vacío controlado tanto antes como después de la cabeza lectora/grabadora, para evitar tirones que podrían romper la cinta), la enrollaba en el carrete de la derecha. Después se pulsaba el inevitable botón de "Start", se hacía el vacío, y la unidad buscaba el comienzo legible de la cinta magnética. Una vez encontrado, quedaba en "Ready" hasta que el programa diera la instrucción de lectura o escritura pertinente. Y sí, el indicador de "Ready" era verde. En una cinta de 2400 pies, que era el tamaño normal de las cintas de entonces, de algo menos de treinta centímetros de diámetro, podían caber unos treinta o cuarenta millones de caracteres (40 Mb, vaya), por lo que los ficheros más grandes, como el de Cuentas, Clientes, etc., necesitaban varias cintas (secciones), que debían ser montadas en orden... a mano. En ocasiones, el Operador se equivocaba y montaba la sección tres antes que la dos. Ya podéis imaginar el desaguisado que se montaba. Por cierto, mientras la unidad de cinta rebobinaba la sección para permitir su cambio, el programa (o sea, todo el ordenador) se quedaba tranquilamente parado, esperando que el operador informara por el teletipo que, al fin, la cinta se había cambiado y podía continuar el proceso.

**Impresora.** La joya de la corona de NCR. Una impresora de líneas de última generación que imprimía en papel pijama de 132 caracteres, y lo hacía rapidísimo. Era de metro veinte o metro treinta de altura, y alrededor de un metro de ancho y de profundo. Tenía un rodillo de 132 caracteres de ancho, con todos los caracteres representados, es decir, en cada posición, y a lo largo de la circunferencia del rodillo, estaban todos los caracteres imprimibles, los de entonces, quiero decir: las mayúsculas, los números y unos veinte o veinticinco caracteres especiales, como el punto, la coma, el espacio, el asterisco, los paréntesis, etc. Cuando mirabas al rodillo de frente, veías una fila de 132 Aes, debajo una de 132 Bes, luego de Ces, y así sucesivamente. En el otro lado (y al otro lado del papel, una vez colocado correctamente) estaban los martillos. Si tenía que escribir una F, el martillo esperaba a que pasara la F del rodillo justo en el punto de golpeo, y entonces saltaba y golpeaba al papel, que quedaba marcado con la letra F. Cuando todos los caracteres de la línea habían sido impresos, la impresora saltaba a la línea siguiente, y así. Para que el carácter quedara impreso, hacía falta tinta... que se obtenía de un papel de calco que se colocaba entre el rodillo y el papel: artesanal, sí, pero funcionaba. Y todo ello, a una velocidad muy alta, pues era capaz de imprimir 1.500 líneas por minuto. La coordinación mecánica necesaria para realizar este proceso correctamente era muy notable para la época. Claro que, a cambio, era el dispositivo que más se estropeaba, fácilmente una o dos veces por semana.

**Entrada Primaria del Sistema.** El dispositivo de entrada del Sistema era el lector de tarjetas perforadas. Los programas los leía en fichas perforadas, el Boot (el arranque de la máquina) lo hacía leyendo las instrucciones de tarjetas perforadas, y los trabajos del sistema también los leía de tarjetas perforadas.

**El Arranque del Sistema (el Boot).** Lo de "Boot" viene de "BootStrap", y es lo mismo que el IPL ("Initial Program Load"). De "boot" viene lo de "Botar la máquina" que se aún se escucha de vez en cuando. Para arrancar aquel monstruo de ordenador, se hacían las operaciones que contaré a continuación (porque no bastaba con apretar el botón y esperar, como ahora, no...). El Century-200 tenía un Sistema Operativo tipo "DOS", es decir, "Disk Operating System", lo que quería decir que el Sistema estaba en un disco: el "Disco del Sistema", naturalmente. Recordad que los discos tenían algo más de cuatro megas, así que podéis haceros una idea de la enorme complejidad del Sistema Operativo: todo él cabía en cuatro megas, y aún sobraban dos y media para que las usaras para otras cosas, por ejemplo, tus programas. El Sistema estaba efectivamente en disco, pero no existían cosas como "Bios", "ROM Memory", etc. O sea, una vez encendida la máquina, no había nada en memoria, ni forma de cargar nada automáticamente. Básicamente, tenías un pedazo de hierro. Encendido, pero hierro, al fin. Había que cargar el Sistema de forma manual. Usando, cómo no, un taco de fichas perforadas (no más de ocho o diez, recuerdo), que era lo único que el ordenador podía leer por puro hardware. El procedimiento era el siguiente:

1. En primer lugar, se tomaba el bloque de tarjetas perforadas de arranque del Sistema y se colocaba en la lectora de fichas. El ordenador estaba parado (el switch de "Halt" estaba activado: cuando se activaba, manualmente, el ordenador interrumpía su operación y se quedaba "clavado" en la instrucción que estuviera ejecutando en ese momento).
2. El Operador señalaba en el dial la dirección de carga del programa (era la 00A0, qué cosas más raras se recuerdan al cabo de los años...). Y entonces pulsaba la tecla "Load" de la consola. Esto hacía que, por hardware, se leyera la primera tarjeta del taco y se cargara en la dirección marcada (la 00A0, claro).
3. A continuación (con la misma dirección 00A0 marcada), pulsaba la tecla "Act". Esta tecla llenaba el registro de instrucciones, marcando que la próxima instrucción a ejecutar fuera la 00A0 (la que marcaba el dial). Es decir, en el principio del contenido de la tarjeta que se acababa de cargar. Esta tarjeta (todas las del taco, en realidad) estaba perforada en "multipunch", es decir, tenía más agujeros de los que el código Hollerith permitía para las tarjetas perforadas, pues representaba caracteres binarios, con las instrucciones, en código máquina puro, que debía ejecutar el ordenador. O sea, una Bios de cartón[[2]](#footnote-2).
4. El Operador bajaba el switch de "Halt" y pulsaba la tecla "Compute". Esto hacía que se comenzara a ejecutar el programa contenido en la dirección marcada. El contenido de los ochenta caracteres de esta primera tarjeta tenía el código suficiente como para poder leer el resto de fichas del taco, cargarlas en las direcciones sucesivas (la primera en la 00F0, la segunda en la 0140, y así), y, al detectar el fin del taco, ceder control a esta nueva pieza de código. En esos quinientos o seiscientos bytes cabía todo el código necesario para ir al disco, recuperar el Sistema, cargarlo en la máquina, arrancar los procesos pertinentes, etc. Compacto, ¿no os parece?
5. Cuando terminaba todos estos procesos, el teletipo de la consola emitía un "READY" (con su campanita al final, y en mayúsculas, que no existían las minúsculas), y el ordenador estaba listo para aceptar trabajos. Todo el proceso podía tardar uno o dos minutos.

Cuando un programa "cascaba" (o sea, daba un error fatal, un zapatazo, un abend, terminaba de mala manera...) la única posibilidad de averiguar qué había pasado era emitir un volcado de memoria (un "dump") en hexadecimal, con todo el contenido de la memoria del ordenador, que se imprimía, y te enviaban para averiguar el motivo del "casque"... y casi siempre con prisa, podéis imaginar. Con tu programa a un lado y el volcado de memoria al otro, y varios lápices o bolis de colores empezabas a marcar el contenido de cada campo, pintabas rayas de conexión, veías en qué instrucción había fallado, y así... hasta que, unos minutos o unas horas más tarde, por fin sabías el motivo del error, para corregirlo. Todos nosotros sumábamos y restábamos en hexadecimal casi mejor que en decimal. ¿He dicho ya que no había calculadoras que operaran en hexadecimal? Bueno, ni casi en decimal: eran unos trastos eléctricos grandotes que imprimían sus cálculos en una tira de papel...hoy aún se ven de éstas en algunas oficinas. Ya podéis suponer que la preocupación por salvar árboles no se había puesto aún de moda: consumíamos cantidades ingentes de papel, porque tampoco había otra alternativa, además: toda la comunicación hombre-máquina se hacía en papel (pijama). Otros sistemas tenían ya pantallas (para los operadores exclusivamente); NCR en los 70, no.

Quizá estéis pensando, los que hayáis tenido la paciencia de llegar hasta aquí, que qué se podría hacer con un ordenador con tan poquísima potencia... Pues casi nada: llevar toda la información del Banco. Todas las aplicaciones importantes estaban en perfecto funcionamiento: Clientes, Cuentas Personales, Créditos y Préstamos, Liquidación de Cuentas, Depósitos, Contabilidad, Nómina, Cartera de Efectos, Valores (con sus decenas de sub-aplicaciones, como Bolsa, Arbitrajes, Renta Fija, Abono de Cupón, Custodia, Traspasos, etc.) en fin, todas las aplicaciones de la época estaban informatizadas, y todas las comunicaciones al cliente se imprimían por las fastuosas impresoras que os he contado. Y todo el software era hecho ex profeso para el Banco: eso de comprar software de otros no existía, salvo los propios programas del Sistema Operativo, claro, que venían de serie con el ordenador.

La plantilla de Análisis y Programación de una gran empresa de la época podía componerse de entre treinta y cincuenta personas. Teniendo en cuenta que todas las Aplicaciones importantes funcionaban, y se daba servicio a las nuevas necesidades en un tiempo razonable, la productividad era muy alta, desde luego. No perdíamos mucho tiempo en documentar ni hacer interminables memorandos justificando el coste: nos limitábamos a escribir el software, ponerlo en Producción, formar a los usuarios en su uso, y dar soporte después. No existían las Bases de Datos. Bueno, sí que existían, pero con capacidades muy escasas todavía, y, que yo sepa, no había casi ninguna adecuada para los Sistemas de NCR. Es decir, los ficheros del Banco, donde estaba toda la información sensible eran... eso, ficheros. Secuenciales. Prácticamente todos, en Cinta Magnética, y se les ponía al día mediante un proceso de actualización conocido como "Padre-Hijo", donde se montaba en una unidad de cinta la versión actual del Fichero Maestro, en otro armario, una cinta virgen para acoger la versión siguiente del mismo Fichero, y se ejecutaba un programa de actualización (que solían ser los más complicados de la instalación) que, teniendo en cuenta el fichero de movimientos (o los ficheros de movimientos, en ocasiones se precisaba tener más de uno), generaba la versión actualizada de dicho Fichero Maestro. Se guardaban varias versiones de cada Fichero (entre 4 y 8, según lo importante que fuera), y se guardaban también los ficheros de movimientos de esos días, para poder repetir un proceso que hubiera resultado erróneo por cualquier causa, generalmente porque el programa no hacía bien su trabajo.

El programador, que sabía qué programa tenía que hacer, incluso (casi siempre) lo que tenía que hacer el programa, se ponía a ello. El primer paso era "hacer el mono". O sea, el organigrama[[3]](#footnote-3) (los finos lo llamaban ordinograma, para distinguirlo del Organigrama de la Empresa, aunque éste último no hacía falta: todos sabíamos quién era quién). "Hacer el mono" en realidad era pintar en un papel (no, en muchos papeles) el diagrama de secuencia del programa. En cualquier caso, los saltos de un lado a otro del flujo[[4]](#footnote-4), según las condiciones que se fueran satisfaciendo, se pintaban con flechitas, que tendían a entremezclarse y volar de arriba abajo, de derecha a izquierda y, en realidad, en todas direcciones ... de ahí la denominación de "código spaghetti" para los programas resultantes. Este término (lo de código spaghetti) es claramente peyorativo, pero es que fue inventado muchos años después. En los años setenta ésa era la manera de programar. La única, no había otra. Hasta 1983 u 1984 no empezó a imponerse la "Programación estructurada", pero aún hubo que mantener programas antiguos durante muchos, muchos años.

Una vez hecho el organigrama (más o menos detallado, según la experiencia y habilidad del programador), se comenzaba a codificarlo, es decir, a traducirlo al lenguaje que el ordenador entiende (NEAT/3, Cobol, Fortran o lo que fuera). El programador agarraba un taco de Hojas de Codificación[[5]](#footnote-5) en blanco, y comenzaba a escribir allí el código. Usábamos lápiz casi todos para poder corregir errores, y solíamos tener seis o siete montones de hojas con las diferentes partes del programa que se iban codificando a la vez... y todo ello, bajo la continua consulta al organigrama... ¿He dicho ya que no había pantallas, ni editores de texto, ni nada parecido?

Al cabo de varios días (dependiendo de la complejidad del programa), terminabas tu trabajo de codificación. La complejidad media de los programas que hacíamos entonces era bastante alta, comparada con la de los que se hacen ahora[[6]](#footnote-6). Preferíamos hacer un único programa muy complicado que tres más sencillos.

Con tu programa terminado, enviabas el taco de hojas al Departamento de Grabación, para que las perforaran. O sea, que cada línea que tú habías escrito fuera copiada literalmente a una ficha perforada, con las perforaciones equivalentes al código que tú habías parido (según el código Hollerith, así llamado por el inventor de la tarjeta perforada, Herman Hollerith)[[7]](#footnote-7). En el Departamento de Grabación, donde una pléyade de grabadoras o perforistas (y lo digo en femenino porque en todos los sitios en que he estado, en Grabación había exclusivamente mujeres, no me preguntéis por qué) se ocupaban de recibir documentación de Oficinas, Servicios Centrales, etc. y grabarlas en cinta o tarjeta perforada, según el programa que leyera los datos. Tus hojas de codificación, con palabras extrañas y nombres impronunciables, como debe ser, se ponían en la cola hasta que le tocaba su turno. ...y tardaba.

Siempre tardaba, primero porque tenía prioridad el trabajo diario (la contabilidad debía cerrar cada día, porque si no...), y segundo, porque a las perforistas no les gustaba nada perforar programas. Cientos o miles de fichas perforadas por programa, un trabajo muy feo, y lo digo por experiencia, porque cuando un programa te corría prisa de verdad, te ibas a Grabación, y aprovechabas que alguna perforadora estuviera libre para perforarte tú mismo el programa. Y era feo de verdad. Además, para asegurar la máxima calidad en la grabación, todo el trabajo se verificaba. Esto consistía en volver a poner el taco de tarjetas grabadas en el alimentador de la perforadora, ponerla en "modo verificación" y volver a teclear de nuevo el programa. Ahora la máquina no perforaba, sino que se aseguraba de que lo que tecleabas coincidía con lo ya perforado. Si no era así, avisaba y había que ver dónde estaba el error (en la tarjeta, o en lo que habías tú tecleado en la verificación), y corregirlo. Un tostón, y encima, caro, por lo que en ocasiones no se verificaban los programas.

Bien, han pasado unos días más, y por fin te devuelven un taco de quinientas, o mil o dos mil fichas con tu programa perforadito, y atado con una goma elástica. Más te vale que las gomas con que el taco viene sujeto no se rompan, porque si se cae al suelo y se desordena... te puedes pasar un par de días hasta recolocarlo de nuevo. Las fichas perforadas nuevas (o sea, para perforar) venían en paquetes de 2.500 fichas: cada paquete medía unos cincuenta centímetros de largo por unos veinte de ancho y diez de alto, y pesaba lo menos diez o doce kilos. O sea, un programa de 1.000 fichas, bastante normal en la época, ocupaba un espacio de unos veinte centímetros, pesaba alrededor de cinco kilos, y tardaba en ser leído por la lectora de fichas tres o cuatro minutos... si no se atascaba.

Entonces, envías al Ordenador tu taco de fichas, para compilar el programa. "Compilación" es el proceso por el que el ordenador lee tu programa, verifica que cumple la sintaxis del lenguaje de que se trate, que los campos que has utilizado están correctamente definidos, etc... Si la compilación acaba con "cero errores", es decir, cumple todas las normas y es, aparentemente, un programa de verdad, entonces el compilador traduce tus instrucciones en Cobol (o el lenguaje que sea), al código máquina que de verdad entiende el ordenador (que sólo sabe de ceros y unos, recordad). Una compilación típica en esa época solía durar entre diez y quince minutos, con el ordenador en exclusiva. Así, tu "MOVE ZERO TO CAMPO", se convertía en una instrucción máquina parecida a, digamos, 64002E4C0E002540, que sería la traducción en código máquina de tu estupenda MOVE. Pues bien, repito, mandabas tu programa a compilar a la Sala del Ordenador... y vuelta a la espera. El ordenador era monotarea, es decir, hace una sola cosa a la vez. Si el Operador lo dedica a compilar, no hace otra cosa. Y si está pendiente de ejecutar el Balance de Contabilidad, o el Abono de Cupón del Banco Hispano Americano, o la Liquidación de Intereses de los Depósitos a Plazo, pues no se compila, como es lógico. Las compilaciones tenían normalmente una prioridad, digamos, baja. Después, el ordenador fue multitarea (en realidad, bitarea), por lo que la espera media se redujo de dos o tres días a sólo uno o dos. Así que, si te corría prisa que se compilara, podías hacer dos cosas:

1. Una: Irte un domingo al Banco, y mientras el único Operador que iba los domingos trasteaba en uno de los ordenadores, tú encendías el otro y te lo apropiabas durante unas horas.
2. Dos: revisarte de arriba abajo tu taco gigante de fichas para detectar errores de perforación, o tuyos. Y esto tenía guasa. Pero guasa. Te podías pasar un día entero para encontrar tres o cuatro errores (o treinta o cuarenta, si no habían verificado), que podías arreglar previamente a la primera compilación, eliminando, con suerte, una pasada (y una espera, en consecuencia).

Tras varios intentos, dependiendo de lo torpe que fueras, por fin ha llegado tu programa sin errores de compilación. Ahora, hay que probarlo. Es decir, asegurarse de que el programa, de verdad, hace lo que se supone que debe hacer (y no sólo lo que tú, programador, te crees que debe hacer, o peor, lo que en realidad has programado, que quizá no tiene nada que ver con lo que querías programar...)[[8]](#footnote-8). Para ello, no queda más remedio que ejecutar el programa con los ficheros que luego va a utilizar de verdad (o unos similares de prueba, aunque la verdad es que nadie tenía ficheros de prueba de nada: se usaban los reales, y punto; aún no había ninguna "Ley de Protección de Datos"). Entonces solicitabas la prueba a la Sala del Ordenador, en un papel al efecto, especificando los ficheros que necesitaba tu programa, la salida esperada y si había que contestar algo por consola (muchos programas preguntaban cosas, por ejemplo, "Diga Vd. la fecha de Proceso", y el operador debía contestar en la propia consola la fecha pedida, que tu programa debía validar para que fuera correcta... bueno, deber, debía, pero no siempre lo hacía).

Podéis imaginar que también había que esperar para probar, claro. Por ello, era común hacer una "Prueba en Mesa" antes de probar (incluso podías hacerla durante el proceso de puesta a cero errores de compilación). Esta Prueba en Mesa significaba que tú ibas a ser el ordenador y a ejecutar tu programa instrucción a instrucción, para detectar fallos antes de la costosa prueba de verdad en máquina. Te inventabas, o tomabas de la realidad, unos datos de entrada, y pacientemente te ponías a desentrañar tu spaghetti, anotando: aquí leo el fichero A (y anotabas el contenido del siguiente registro del fichero A en su sitio), ahora comparo si F es mayor que G, si es mayor, salto a Z (y mirabas si lo que tenías apuntado en F era mayor que G, si era así seguías por Z, si no, continuabas), ibas machacando los valores de los campos con los nuevos valores... Y así durante horas. Parecerá increíble, pero era utilísimo, no sólo porque detectabas errores lógicos, sino porque te aprendías el programa de tal manera que, una vez que lo probabas de verdad, o sea, en máquina, y no funcionaba como debía (o sea, lo normal), casi ibas derecho al sitio donde estaba el error. Es una sensación muy curiosa, que yo he tenido muchas veces y que no sé explicar, la de "intuir" en que parte exacta del código está el error... y acertar.

Bueno, al fin ibas a probar. Generalmente te avisaban para hacer la prueba, ya que muchas veces, sólo viendo cómo se encendían y apagaban las luces de la consola o los dispositivos, o cómo se movían las cintas, sabías si había algún problema (igual esto no os lo creéis, pero prometo que es cierto). O más importante, porque estando allí, eras capaz de sacar mucho más partido a la prueba, que como habéis visto, era realmente costosa de hacer. Me explico con un ejemplo: era bastante corriente que en la primera prueba el programa se metiera en un bucle (...infinito, claro). Las cintas se paraban, los discos no se movían, y las luces se quedaban encendidas con un patrón determinado. Un bucle, sí, ya (como siempre, más bien) pero... ¿dónde? Si estabas allí, podías hacer lo siguiente:

1. Uno: Parar el ordenador (ya os conté que tenía un switch, una palanquita de "Halt" que, cuando la movías, dejaba al ordenador parado en la instrucción donde estuviese).
2. Dos: Mirar las luces de la consola que te indicaban la dirección física de memoria donde estaba la instrucción que se estaba ejecutando. Por ejemplo, la 3E58.
3. Tres: Ibas a tu programa y mirabas (en la parte del listado de la compilación donde tenías la conversión de tu código fuente a código máquina) dónde caía la dichosa dirección 3E58, que estaría en un cierta parte del programa, por ejemplo donde estás calculando alguna cosa.
4. Cuatro: Si no quedaba claro qué podía estar pasando (muchas veces, con sólo saber que había un bucle en la zona, ya encontrabas el error), podías ir ejecutando instrucción a instrucción, pulsando sucesivas veces la tecla "Compute", y observando el flujo real que seguía el programa. Podías también averiguar el contenido de ciertos campos importantes para el propio control del bucle.

Si con esto no veías qué pasaba, cancelabas la prueba, te retirabas a tu mesa y allí te ponías a revisar de arriba abajo el programa hasta que encontrabas el error. Pero era común que encontraras inmediatamente lo que pasaba (por ejemplo, habías escrito IF A = B, y tenía que ser IF A NOT = B). Entonces sí que podías continuar con la prueba, para aprovecharla al máximo. Para ello, ibas al código máquina donde estaba esa instrucción errónea concreta y machacabas (a base de usar la consola de forma parecida a la que conté anteriormente) el código de instrucción (por ejemplo, sustituías un código "EA" -saltar por igual- por un "ED" -saltar por distinto-). Y ya podías continuar la prueba como si hubieras cambiado el código (que habría que cambiar en el programa final en cualquier caso). Con los debuggers que existen ahora, esto puede parecer (y es, qué demonios) antediluviano, pero es que ése era el debugger de la época... y había que conocer la máquina de-pe-a-pa para poder utilizarlo.

Si tu prueba había fallado, lo que era (y sigue siendo) lo normal, te llevabas los listados de los ficheros de entrada y de salida a tu mesa, para allí determinar dónde estaba el error, y cuando lo localizabas, volvía a empezar el ciclo perforación-compilación-prueba, hasta que sorprendentemente... ¡El programa funcionaba correctamente! (o al menos, eso parecía, que no es lo mismo). Lo das por válido y das las instrucciones a Producción para que se ejecute cuando le toque. Ahora sí hacías un diagrama precioso de tu programa, usando la plantilla, para que quedara claro, indicando entradas y salidas, datos por consola, y qué hacer en caso de error (llamar al programador). ...Y al fin, el programa entra en Producción, y por tanto, pasa a Mantenimiento. El taco de fichas de varios kilos que representa el programa definitivo lo almacenas amorosamente en unas gavetas especiales donde se encuentran los programas en Producción. Algún día habrá que modificarlo para meter algún cambio, así que más vale que esté a buen recaudo. El listado de la compilación definitiva lo encuadernas con unos artilugios especiales que había para tal fin, y lo almacenas, no menos amorosamente, en un armario especial para guardar listados que entonces se vendía (y que supongo que ahora, no). En cuanto a la documentación (organigrama, servilleta de bar, apuntes, resultados de la prueba, etc.), directamente la tiras... Total, lo más probable es que el programa que por fin ha quedado como definitivo no se parezca mucho a los planes iniciales.

# 1980

A principios de los ochenta aproveché una buena oferta y salté a otro Banco Nacional. El cambio supuso una mejora económica (pequeña) y un nuevo y apasionante proyecto. El ordenador de este nuevo Banco (nuevo para mí, quiero decir) era un IBM 3081 nuevecito, con dos procesadores (¡nada menos!) y 16 Mb de memoria. O sea, un gran mainframe de IBM, de lo más que había en la época (casi casi lo único que había como ordenador central). Este ordenador era evolución de la serie IBM 303x, que a su vez lo era del mítico IBM 370, que había sustituido en los primeros setenta al no menos mítico IBM 360. Su Sistema Operativo era MVS, y tenía Bases de Datos IMS (DL/1) y gestor de teleproceso IMS/DC, ambos recién estrenados, y que iban a dar servicio al nuevo Sistema Integrado Online de dicho Banco (que había que construir desde cero, claro). Se programaba en Cobol (salvo ciertos módulos de Sistema, que se escribían en Assembler), es decir, todo el software de Aplicación estaba programado en Cobol. Es más, en estos tiempos (que yo sepa) sigue programado en Cobol. Y, en una buena parte, siguen siendo los mismos programas, remozados, cambiados, modernizados... pero los mismos. Veréis que esto no es tan extraño como parece: aunque con potencia infinitamente superior, los z/Series de hoy se parecen mucho, de cara a sus programadores y usuarios, al venerable IBM/370 de hace casi cuarenta años.

Entonces, lo primero que hay que mencionar es la estabilidad de esta gama de ordenadores. Se han producido, y se siguen produciendo, gigantescos avances en ellos igual que en el resto de ramas de la informática, pero, al estar todos ellos sin excepción dedicados a operaciones críticas, la estabilidad es la primera máxima de diseño. Eso quiere decir, por ejemplo, que un programa escrito y compilado en los años setenta sigue funcionando hoy en día en los ordenadores actuales... ¡sin necesidad siquiera de compilarse de nuevo con cada cambio de versión! El código máquina de estos ordenadores es hoy el mismo que hace cuarenta años. Naturalmente, los procesadores son muchíiisimo más complejos ahora, hay muchos más códigos de instrucción para hacer cosas cada vez más sofisticadas... pero los códigos originales se mantienen. Es decir, hay una compatibilidad hacia delante que, simplemente, no existe en ninguna otra gama de ordenador, de la marca que sea. Los mainframes actuales se denominan IBM z/Series, y no son más que la evolución de los 360, 370, 303x, 308x, 3090, 390, etc.

La principal diferencia es, además de su potencia, de varios órdenes de magnitud superior, el tamaño: el 3081 con que operaba aquél banco, más todo sus dispositivos (discos, cintas magnéticas, impresoras, unidades de comunicaciones, etc.) atestaba él solito la Sala de Ordenadores entera, con sus doscientos cincuenta o trescientos metros cuadrados... y ahora en ese espacio cabrían decenas de mainframes. Son máquinas realmente potentes (los datos que cito a continuación son del z10Ec, el más potente de los actuales mainframes de IBM[[9]](#footnote-9)):

* Hasta 64 procesadores de alta capacidad por ordenador. Son procesadores de cuatro núcleos a 4,4 Ghz de reloj. Esto significa un ciclo de reloj de 0,22 nanosegundos, es decir, 220 picosegundos. Con una configuración "normal", de 16 ó 24 procesadores (en España no creo que haya ningún mainframe con 64), es capaz de procesar algunos miles de MIPS (Millones de Instrucciones por Segundo: Millones reales (y verificables) de Instrucciones cada Segundo, no Millones teóricos, que es lo que se acostumbra a medir en otros Sistemas.
* Hasta 1,5 Tb de memoria de alta velocidad (eso es Mucha memoria...).
* Hasta 1024 Canales de Entrada/Salida. Son canales de Alta Capacidad y velocidades de transferencia muy elevadas, de varios Gb por segundo.

Y sin embargo, viendo las características físicas no parece tan potente, comparado con máquinas especializadas de procesamiento en paralelo, como la propia "Blue Gene" de IBM, que tienen miles y miles de procesadores. Y es que la diferencia está en su Software. Su Sistema Operativo por excelencia es, hablando con propiedad, el único Sistema Operativo que existe (se trata del antiguo MVS, que tras varios cambios de nombre ahora se llama z/OS; IBM ofrecía también por aquella época, en los años 70 y 80, el DOS/VSE, igual que ahora también ofrece Linux para z/Series, ignoro con qué éxito comercial). Cuando se enumeran las funciones y características que, teóricamente, debe tener todo Sistema Operativo, sólo MVS (perdón, z/OS; es que todos los que hemos trabajado con él le seguimos llamando "MVS") cumple todas ellas. Por ejemplo:

1. Disponibilidad total. IBM dice en su marketing que tiene un 99,999% de disponibilidad, y no sólo es cierto, sino que yo creo que es un 100%. De hecho, los mainframes de IBM sólo suelen apagarse una vez al año (generalmente en Navidades, Viernes Santo, o así), más bien por el prurito de hacer un Arranque en Frío al menos una vez al año (o sea, apagar y encender, vaya) y así limpiar la memoria de posibles zonas muertas, que porque haga realmente falta.
2. Independencia del Sistema Operativo de las Aplicaciones (incluso las propias del Sistema). Es decir, absolutamente todos los productos, sean cuales fueren, se pueden instalar en caliente, sin necesidad de apagar y encender nada. Esto reza incluso para el propio Sistema Operativo: se puede subir de versión sin necesidad de pararlo. Y desde luego, para todos los productos, por críticos que resulten: IMS, CICS, DB2, RACF, VTAM, etc. Y, por descontado, el MVS (perdón otra vez, el z/OS, la costumbre...) no se cae nunca. Al menos, yo no lo he visto en años.
3. Absoluta garantía de que una partición no puede acceder a datos de otra, ni por error, ni a posta. Además, no hay virus (al menos, que yo conozca) para mainframes: la seguridad es máxima[[10]](#footnote-10).
4. Capacidad de configurar varias máquinas lógicas (particiones) dentro de una única máquina física. Con ello se puede trabajar como si tuviéramos tres o cuatro máquinas diferentes, más pequeñas, pero que se comportan a todos los efectos como si fueran máquinas físicas diferentes. Además, la partición se realiza no distribuyendo partes físicas de la máquina (canales o procesadores, por ejemplo) sino por MIPS. Podemos definir una máquina de 150 MIPS, otra de 400 MIPS y otra de 700 MIPS, por ejemplo. Y, naturalmente, se puede reconfigurar los tamaños de cada partición en cualquier momento... y sin parar nada, claro.
5. Rendimiento combinado inalcanzable para cualquier otra máquina. Por ejemplo, un mainframe típico (no el más grande, repito) es capaz de dar servicio él solo a:

* 500 ó 600 trabajos batch[[11]](#footnote-11) (en la jerga "mainframera", iniciadores), donde se están ejecutando trabajos batch: liquidaciones, extractos de cuentas, abonos de dividendo, tareas de backup, pruebas de programas, etc., todo lo que no necesite hacerse online. No 500 ó 600 al día, no. 500 ó 600 a la vez, continuamente, todo el día sin parar, todo el año sin parar.
* Varios gestores de teleproceso diferentes (IMS ó CICS), sirviendo cada uno algunos millones de transacciones online diarias. Para que os hagáis una idea: un Banco nacional típico puede tener quizá diez o quince millones de transacciones online diarias, una operadora de Telecomunicaciones, quizá registre 80 millones de llamadas diarias, etc.: se procesa online muchísima información hoy en día. ¿Os hacéis una idea de la tasa de transacciones por segundo que supone esta carga? 200, 300, 400 transacciones por segundo, cada segundo, todos los segundos. Son realmente muchas. Estas transacciones, habitualmente sub-segundo, pueden ser servidas por uno sólo de estos gestores, por ejemplo, un IMS. Y suele haber otros gestores de Teleproceso dedicados a otras cosas, como la Contabilidad, los Seguros, etc.
* Varios centenares de particiones de time-sharing, el famoso TSO, (donde cada usuario ve la máquina como si fuera literalmente para él solo). Por ejemplo, programadores editando y compilando programas, usuarios finales lanzando peticiones de información, operadores gestionando el sistema, etc. Con tiempos de respuesta generalmente sub-segundo también.

...SIMULTÁNEAMENTE. Es decir, todo lo anterior, a la vez (al mismo tiempo). Un mainframe de IBM está habitualmente usando su CPU (todas ellas) al 100%. Horas y horas, días y días, meses y meses, sin parar. Es normal verlo incluso al 102% (cosa aparentemente imposible, debida al pre-fetch de instrucciones, que permite ganar algo de tiempo al solapar la ejecución de instrucciones consecutivas). Y no pasa nada. Muchos otros sistemas operativos, al llegar al 80% de uso de CPU comienzan a dar señales de lentitud. Y a partir del 90%, se vuelven inestables. El z/OS, no. Es más, casi todos los que trabajamos normalmente con ellos "sabemos" que cuando están a menos del 90%, están "vagos", como si les costara más de lo normal despachar los diferentes trabajos... Vale, ya lo sé, es una sensación que no tiene base alguna, pero es que la tenemos mucha gente...

Aunque son sistemas muy fiables por su diseño, también en los mainframes se producen errores y roturas de hardware, como es natural. Con 12 o 16 procesadores que funcionan día y noche, es posible que, por muy fiables que sean, tarde o temprano uno de ellos falle y tenga que ser reemplazado, y lo mismo ocurre con el resto de componentes: memoria, discos magnéticos, unidades de comunicaciones, etc. Imaginemos, por ejemplo, una moderna instalación, que puede tener del orden de 150 Tb de espacio en disco disponible (no sólo se necesita para guardar la información permanente, sino para espacio de trabajo, copias, etc.). Dependiendo de en qué momento se fueron incorporando los diferentes discos a la instalación, estos tendrán una tecnología u otra, girarán más o menos rápido, y su capacidad individual estará más o menos entre 50 y 500 Gb. Supongamos una media de 150 Gb por disco, lo que puede ser una cifra bastante real. Pues bien, en esta instalación prototipo, sólo discos de mainframe hay unos 1.000, más o menos antiguos, girando 24 horas diarias a un mínimo de 7200 vueltas por minuto. Supongamos también que estos discos tienen un MTBF (Tiempo Medio Entre Fallos, una medida de la fiabilidad de un componente) de 200.000 horas, que no está nada mal. O sea, si tenemos un solo disco, podemos esperar, si tenemos suerte, que falle de media cada 200.000 horas, que son... ¡veintitrés años! Pero es que... ¡tenemos 1.000 discos, o más! Sin hacer muchas cuentas complicadas, podemos darnos cuenta de que en realidad podemos esperar un fallo de un disco individual cada quizá 150 ó 200 horas, o sea, tendremos un disco roto cada par de semanas o así. Y puede fallar cualquiera, desde el menos importante, que sólo sirva como espacio de trabajo, al propio disco donde se encuentra el Sistema Operativo. Pues... la instalación no puede pararse por un disco roto, sea cual sea el que se rompa; y lo mismo si lo que falla es cualquier otro componente, por crítico que sea. En los mainframes es poco menos que imposible encontrar un componente aislado cuya rotura origine una parada total del sistema, y toda sustitución de un elemento hardware estropeado se realiza sin necesidad de parar el resto de los componentes (ni los iguales, ni otros distintos) ni, desde luego, el Sistema Operativo, que simplemente marcará ese dispositivo como inutilizable mientras dure la avería, y volverá a utilizarlo cuando ésta se subsane. Aunque no es propiamente un Sistema "Fault Tolerant", se trata de un sistema realmente fiable.

En fin, la gran mayoría de grandes Cajas y Bancos, los diferentes Organismos de la Administración (Hacienda, Trabajo y Seguridad Social, Interior, etc.), las grandes compañías industriales y comerciales, sobre todo las que llevan bastantes años en el mercado, casi todas ellas continúan usando mainframes de IBM para sus Operaciones, y que yo sepa, ninguna tiene la menor intención de cambiar a otro tipo de máquinas. Porque, además, resulta un Sistema muy barato de administrar. Con una veintena de Técnicos de Sistemas (muy bien pagados, eso sí) se puede gestionar el parque de mainframes de una gran empresa... cuando con toda seguridad la misma empresa necesitaría cientos de ellos para administrar un Sistema de Información similar basado en, por ejemplo, UNIX.

Ojo, no me malinterpretéis, yo no he dicho que un mainframe de IBM sea una máquina barata: en absoluto lo es. Su coste es muy superior, sobre el papel, al de un Sistema UNIX o Linux equivalente, y no digamos Windows... Agravado por el hecho de que los programas imprescindibles (Sistema, Base de Datos, Gestor de Teleproceso, Gestor de Seguridad...) IBM no los vende, sino que son alquilados: no se adquiere una licencia permanente de uso (la que permite utilizar de por vida el producto mientras no se actualice... y mientras siga funcionando, claro), sino sólo el derecho a usar el software durante un cierto tiempo, generalmente un año, incluyendo el mantenimiento, nuevas versiones, solución de problemas, etc. Y no se trata de productos baratos, precisamente. Pero si nuestra empresa no se puede arriesgar a que una simple actualización de, digamos, el software de la impresora, te deje frita la Aplicación de Cuentas Corrientes, o la Aplicación de Pedidos... es la mejor opción.

El otro cuasi-desconocido (y, en este caso, incluso denostado) protagonista del mundo de la informática es el Lenguaje COBOL. Porque en 1980, igual que ahora, el lenguaje por antonomasia en que se programaban los mainframes de IBM es, sobre todo, Cobol. Ya desde el principio, Cobol define muy claramente la estructura que debe tener un programa, mediante una organización en Divisiones, éstas en Secciones, y éstas, a su vez, en Párrafos, en contraposición al otro gran lenguaje de alto nivel de la época, el Fortran, donde puedes definir cualquier cosa en cualquier punto, cosa que a los ingenieros seguro que les venía bien, pero no cuando se escriben programas de gestión. Hay cuatro divisiones, que son, siempre en este orden:

* IDENTIFICATION DIVISION, donde se pone la información de identificación del programa: Su nombre, su autor (o autores), Fecha de escritura, etc. Esta División sirve más bien a título de comentario.
* ENVIRONMENT DIVISION, donde se especifica la información de entrada/salida: Qué ficheros usará el programa, cómo son (secuenciales, indexados, etc), cómo se llamarán para el programa y, muy importante, cómo se llaman de verdad en el Sistema, etc.
* DATA DIVISION, donde se definen... los datos, sí. Los registros de los ficheros (en la FILE SECTION), las áreas de trabajo (en la nunca bien ponderada WORKING-STORAGE SECTION), y, en caso de tratarse de un módulo (que es llamado por otro programa de mayor rango), las áreas de intercambio de información (en la LINKAGE SECTION). La DATA DIVISION es el reino de la PICTURE, la cláusula de definición de los datos más potente jamás inventada.
* PROCEDURE DIVISION, donde se define lo que es el programa en sí, las instrucciones que conformarán tu programa, con sus lecturas y escrituras, movimientos de campos, cálculos, ejecuciones de rutinas, etc., hasta alcanzar el STOP RUN, donde el programa termina tranquilamente... si antes no ha dado un cascotazo, claro.

Y es emblemático en Cobol el uso del punto para indicar el fin de la frase, es decir, el fin de sentencia o del párrafo, de la misma forma que el punto se usa en la escritura para terminar una frase o párrafo. La principal ventaja de usar Cobol reside en su capacidad de auto-documentación: escribir un programa en Cobol es prácticamente lo mismo que describir lo que hay que hacer (en inglés). Por ejemplo:

**READ INPUT-FILE INTO REGISTRO-ENTRADA AT END MOVE 'SI' TO INDICADOR-FIN-FICHERO.**

**PERFORM PROC-FACTURA UNTIL COD-FACTURA NOT EQUAL TO COD-FACTURA-ANT.**

**EXAMINE CAMPO-ALFANUMERICO REPLACING ALL 'A' BY SPACES.**

**ADD 1 TO CONTADOR-REGISTROS.**

**CALL 'MODULO' USING ARGUMENTO-1, ARGUMENTO-2.**

**IF INDICE-TABLA GREATER 1000 DISPLAY '\*\*\*LA TABLA SE HA EXCEDIDO'.**

**MULTIPLY NUM-UNIDADES BY PVP-UNITARIO GIVING PVP-TOTAL ON SIZE ERROR GO TO PVP-EXCEDIDO.**

**WRITE OUTPUT-REGISTER FROM REGISTRO-DE-SALIDA.**

**PERFORM HACER-CALCULO THRU FIN-HACER-CALCULO.**

**GO TO LEER-FICHERO.**

Cobol es un lenguaje realmente "verbose": seguramente para hacer las mismas tonterías de las sentencias anteriores, en C (con o sin los signos de la suma) o en Java se requerirían quizá tres líneas cortitas en total... pero esa "verbosidad" redunda en un buen entendimiento de lo que hace un programa Cobol cuando lo mira un programador distinto de aquél que lo escribió... veinte años después. Y eso, en una instalación informática de verdad, con decenas de miles de programas, y millones de líneas de código en Producción, cotiza muy, pero que muy caro.

Ya he comentado que muchas Aplicaciones llevan veinte años o más en Producción, y es lógico pensar que han debido sufrir cambios durante este tiempo: naturalmente que es así. Cambios legales, nuevas necesidades de la compañía, nuevos productos, etc., obligan a cambiar funcionalidades, ficheros, bases de datos... y todo cambio termina obligando a cambiar a su vez uno o varios programas. O muchos. O todos. En estas condiciones, poder entender rápidamente la funcionalidad de un programa, localizar el lugar donde debe ser modificado, modificarlo, y que funcione correctamente según las nuevas especificaciones, y todo en un tiempo razonable, es realmente el factor clave en la elección de un lenguaje de programación. Y en esto, Cobol se lleva la palma.

Me da a mí la sensación, tras hablar con muchos colegas, de que el paradigma sobre el mantenimiento de las aplicaciones escritas en C, por ejemplo, es que resulta más barato reescribir el código de nuevo, cuando hay que hacer una modificación de cierta entidad, que tocar el código existente... porque casi siempre el programa resulta prácticamente incomprensible para un programador distinto del que lo escribió (y en ocasiones, incluso para el mismo que lo escribió). Esto no se puede tolerar en una gran instalación. Que me perdonen los fans del C, o del Java, o del Perl, o del propio Ada, pero esta es mi opinión... y la de muchos otros, también.

Como Gestor de Bases de Datos, se usaba IMS/DB, también conocido como DL/1 (aunque en realidad DL/1 (por "Data Language/1") era el Lenguaje de manipulación de la información en la Base de Datos). IMS/DB es (aún quedan por ahí aplicaciones escritas en DL/1) una Base de Datos jerárquica, es decir, su diseño toma forma arborescente, donde cada registro (Segmento, en su terminología) puede tener "padres" (de los que hereda sus claves), "hijos" (a los que cede sus claves) o "hermanos" (otros hijos del mismo padre, que están físicamente ordenados entre sí por algún criterio o clave). Esta Base de Datos no es sencilla ni de diseñar ni de programar correctamente, pero una vez hecho tiene un rendimiento extraordinario, y fue capaz de sacar un excelente partido a los ordenadores de la época (de limitadísimas capacidades si lo comparamos con cualquier ordenador actual, incluyendo el que tú, lector, tienes encima de la mesa).

En el mercado había alguna otra Base de Datos por entonces, todas jerárquicas o en red, pero IMS fue la que mayor éxito comercial tuvo, sobre todo en los ordenadores de IBM, por motivos obvios.

Por cierto, sé que la mayoría de vosotros, sufridos lectores, estáis convencidos de que las Bases de Datos Jerárquicas han pasado a la historia, fagocitadas por las hoy omnipresentes Bases de Datos Relacionales, pero quizá cambiéis vuestra opinión cuando penséis que el hoy tan usado lenguaje XML es, en realidad, puramente jerárquico. Para almacenar información descrita en XML (información en serio, quiero decir) es mucho más eficaz utilizar una Base de Datos Jerárquica, que no una Base de Datos Relacional, donde para plasmar la Jerarquía, se precisan tablas de relación entre tablas, lo que hace la solución mucho menos eficaz: para recuperar una rama completa del árbol es preciso hacer join tras join... y los joins no suelen ir muy rápido, precisamente. Cuando tenemos miles de registros, no importa demasiado, pero si tenemos almacenados decenas o cientos de millones, entonces sí que importa.[[12]](#footnote-12)

Bien, ahora que he descrito sucintamente el paisaje, voy a contar a qué se dedicaba el paisanaje, es decir, cómo diseñábamos, programábamos, probábamos y poníamos en Producción las Aplicaciones en este entorno. Los Analistas Funcionales (ya no necesariamente "Jefes de Proyecto"), casi todos del Departamento de Organización y Métodos, escribieron el Análisis Funcional de las distintas Aplicaciones: Cuentas Personales, Contabilidad, Valores, etc. No había directrices claras sobre cómo escribir estos documentos, así que cada analista lo escribía como sabía y como podía. Ahora lo llamaríamos más bien un "Análisis de Requerimientos". A veces eran muy detallados en la descripción de la operación elemental, y a veces terriblemente ambiguos (claro que ahora sigue pasando lo mismo). La colección de estos "Análisis Funcionales", junto con las especificaciones de la solución técnica formaba el "Plan de Sistemas", piedra angular a la que todos nos referíamos cuando había discrepancias. Que las había. Terminado el Análisis Funcional/de Requerimientos, se realizaba el Diseño Técnico de la Aplicación. Esto consistía en las siguientes tareas:

1. Diseñar las Bases de Datos IMS de la Aplicación. Un punto crítico, porque un fallo en el diseño inicial podría costar muchísimo tiempo y esfuerzo (y por tanto, dinero) en corregirlo, una vez terminados los programas. Gastábamos muchísima saliva en interminables reuniones para determinar, entre todos, el mejor diseño posible teniendo en cuenta los requerimientos, el número de transacciones esperadas, etc. Y tengo que decir que al final casi siempre llegábamos a un consenso de diseño que, en general, era el menos malo de los posibles (porque ya sabemos que el mejor, lo que se dice el mejor, no existe...).
2. Diseñar las transacciones, es decir, qué datos deben viajar entre el terminal financiero (el famoso IBM 3600) y el Ordenador Central, para minimizar el tráfico, pero que no falte ningún dato necesario.
3. Repartir las funciones de la transacción: qué debía hacerse en el terminal financiero y qué en el mainframe.
4. Diseñar el batch. Muy importante, porque las transacciones capturan la información, la validan y toman decisiones inmediatas, pero toda la consolidación se hacía (y en buena parte, se sigue haciendo hoy en día) en batch, en un proceso por lotes[[13]](#footnote-13). Muchísimos procesos son netamente batch: Liquidar cuentas, Emitir Recibos, Imprimir Extractos Mensuales, Hacer el Balance de Contabilidad, Abonar Dividendos, etc., son procesos costosos que necesitan acceder a muchísima información, y donde no hay nadie esperando respuesta al otro lado del terminal. Todo esto se hace en batch.
5. Diseñar los módulos comunes. Son piezas de software reutilizables en diversas partes de una Aplicación, que se aíslan de antemano y se programan una única vez, asegurándose de que hacen perfectamente la función definida, y con la máxima eficiencia posible[[14]](#footnote-14). Los había genéricos (como, por ejemplo, dadas dos fechas, calcular la diferencia en días entre ellas, esto se usa muchas veces en los procesos bancarios), que utilizaban todas las Aplicaciones, y otras específicas de cada Aplicación (como el módulo de Contabilización de Apuntes, que sabía a qué cuenta contable se refería cada tipo de movimiento y realizaba correctamente el apunte en la Base de Datos de Contabilidad).
6. Diseñar los programas de interfase con las Aplicaciones Antiguas. Importantísimo: Naturalmente, las Aplicaciones no se arrancaron todas a la vez; de hecho cada Aplicación tampoco se arrancaba en todas las oficinas a la vez (técnica que, por lo que yo sé, se sigue usando hoy en día). Entonces, al arrancar Cuentas Corrientes en un par de oficinas, deberían sus datos integrarse con los del resto de Oficinas (que seguían trabajando con las Aplicaciones Viejas), hasta que todas las oficinas estuvieran migradas a la nueva Aplicación. Y, en cualquier caso, esta Aplicación de Cuentas Corrientes debía comunicarse con las de Contabilidad, Valores, etc., que seguían siendo las antiguas, hasta que pudieran ser sustituidas, meses o años después. Pregunta retórica a aquellos de vosotros que estáis estudiando informática: ¿Alguien os ha contado estas cosas? Si la respuesta es Sí, podéis felicitaros, a vosotros mismos, y a vuestros profesores.

Una vez terminado este trabajo, fundamental para que todo vaya luego bien y, sobre todo para que la Aplicación resultante sea mantenible de cara al futuro, había que realizar las especificaciones de los programas individuales, para su programación. Éste fue, mayormente el trabajo que comencé haciendo allí, como "Analista Orgánico", participando también en el Diseño Técnico de la etapa anterior. Para hacer el Análisis Orgánico de los Programas (tanto los online –transacciones- como los programas batch), realizábamos los "Cuadernos de Carga" donde el Analista describía las principales funciones que debía realizar el programa. A lápiz, oiga. Para poder borrar, naturalmente. Preparamos unas hojas standard preimpresas, donde dibujábamos (con la plantilla, esta vez sí) el diagrama del programa (Bases de Datos a las que accedía, Ficheros que leía/escribía, Impresos que generaba, etc.). Después, en román paladino[[15]](#footnote-15), describíamos con cierto detalle lo que tenía que hacer el programa de marras: Que si leer los datos de la transacción de la cola de entrada; que si validar de tal manera los datos de entrada; que si comprobar si había saldo o no en la cuenta de tal otra; qué hacer si no había saldo, o si la cuenta estaba bloqueada; cómo actualizar los saldos y los movimientos, qué responder y de qué manera a la oficina... esas indicaciones que permitían al programador escribir el programa y que casi hiciera todo lo que debía hacer...

Las transacciones online sólo podían acceder a las Bases de Datos IMS, que estaban siempre en disco, pero los ficheros Maestros seguían estando mayormente en cinta magnética (los discos seguían sin tener capacidad para guardar allá toda la información necesaria...). Por lo tanto, para actualizar estos ficheros en cinta, mediante cadenas batch (compuestas de una serie consecutiva y secuencial de programas tuyos y programas de utilidad, sobre todo el omnipresente Sort[[16]](#footnote-16)), se utilizaba un tipo de proceso de actualización denominado "Padre-Hijo", y que sigue siendo muy útil hoy en día. En el otro lado de la línea estaba la misma transacción, pero vista desde el terminal financiero. Había, pues que hacer otro Cuaderno de Carga (por otro Analista diferente y especializado en este Sistema Financiero), coordinándose con el del mainframe, para tratar la misma transacción, especificando aquí lo que debía hacer en el front-end, es decir, en la parte que "da la cara" ante el cliente: Definir qué campos son obligatorios y cuáles opcionales para esta transacción, cómo validarlos, cómo rellenar el mensaje para la central, enviarlo...(esperar respuesta[[17]](#footnote-17))..., cómo procesar la información recibida del Ordenador Central, qué imprimir y dónde, etc. Los programadores recibían los cuadernos de carga una vez terminados, y comenzaban entonces su trabajo: codificar, compilar y probar sus programas. Inicialmente usábamos la misma técnica que la descrita para los NCR Century 200.

# 1984

Al principio seguimos utilizando fichas perforadas (la costumbre es la costumbre), pero las cosas cambiaron muy rápidamente. Una de las características más importantes del MVS era su TSO, subsistema de tiempo compartido que permitía, usando un interfaz de línea de comandos por pantalla (¡no en ficha perforada, sino tecleando!), acceder simultáneamente a diferentes usuarios como si todo el mainframe fuera sólo suyo... cosa que no era cierta, claro, pero lo parecía. Y rápidamente se instaló también, sobre el TSO, el ISPF que permitía, ahora sí, un acceso mediante pantalla completa, y un magnífico editor de programas (para qué mentir: el mejor que yo conozco... y mira que conozco), y que revolucionó la informática y la forma de trabajar, sobre todo de programadores y operadores (porque nosotros, los pobres Analistas, seguíamos escribiendo páginas y páginas como tontos[[18]](#footnote-18)).

Eran pantallas monocromas (de fósforo, en verde o en amarillo, según), dedicadas, conectadas con cable coaxial al concentrador de comunicaciones que, en definitiva, las conectaba con la IBM 3705, de veinticuatro líneas de ochenta caracteres[[19]](#footnote-19), y desde luego que su interfaz no era gráfico, sino de caracteres (y sólo las mayúsculas) y a base de teclas de función (PF8, adelante; PF7, atrás; PF3, grabar y salir; PF5, buscar, y así...

O sea, que ahora, una vez leído por primera vez el taco de fichas, el programa quedaba almacenado en una librería de programas fuente (cada programador tenía la suya propia, que gestionaba a su conveniencia, almacenada permanentemente en disco, aunque todos podíamos acceder en caso de necesidad a las librerías de los demás), y entonces, las modificaciones al programa para compilarlo hasta ponerlo a cero errores (o para conseguir que funcionara durante las pruebas), se hacían usando el editor del ISPF, y mediante pantalla (IBM 3270).

Rápidamente, dejaron de perforarse los programas (con enorme alivio por parte de las pobres perforistas), y los programas se escribían ya completos en la pantalla, reutilizando partes de código similares de otros programas, inveterada costumbre de todo buen programador en Cobol. Cómo será el asunto, que hay malas lenguas que afirman que no es verdad que haya en el mundo millones de programas distintos escritos en Cobol, sino más bien millones de versiones distintas del mismo programa.

Ya sabéis que en los cincuentas, sesentas y primeros setentas, los informáticos de la época (en muchos casos, ingenieros metidos a informáticos), programaban sus rutinas como buenamente sabían y podían. Fueron inventando las diversas maneras de programar conforme programaban, con lo que los más avispados (o afortunados, quién sabe) fueron encontrando maneras de solucionar problemas diversos sobre la marcha. Pero, en definitiva, la forma aceptada de programar era, mayormente, a la "cada maestrillo tiene su librillo". Claro que, si nos damos cuenta, el problema no era todavía excesivamente importante: Había pocos ordenadores funcionando, y no había muchos programas en Producción, y, sobre todo, éstos eran muy pequeños. No había otro remedio, con 8, 16 ó 32Kb de memoria, no se podían hacer programas muy complejos, simplemente porque no cabían. Y sin embargo, alguien estaba ya dándose cuenta de que esa no era la manera correcta de programar de cara al futuro[[20]](#footnote-20). Cuando el uso de las máquinas se fuera generalizando, éstas fueran cada vez más y más potentes, y más y más programadores se fueran incorporando a la profesión, los programas spaghetti no serían adecuados, debido no sólo a la dificultad de diseñarlos, programarlos y probarlos, sino sobre todo, de mantenerlos. Conforme más y más aplicaciones se ponían en marcha, más y más aplicaciones había que mantener, modificar, cambiar... y la programación "personal" constituía un problema serio, sobre todo cuando el autor original del engendro (digo, del programa) ya no estaba en la empresa para desentrañar lo que ahí dentro había (...y a veces, ni así).

En 1966, Bohm y Jacopini publicaron un artículo en el que demostraron formalmente que, usando exclusivamente tres estructuras de control era posible programar cualquier función computable. Las tres estructuras eran, claro está, la secuencial, la repetitiva y la alternativa. Combinándolas recursivamente es factible realizar cualquier programa. Las tres estructuras básicas: Secuencia, Iteración y Selección.

* Una Secuencia consiste en una serie de elementos que ocurren secuencialmente, uno detrás de otro, y siempre en el mismo orden. Esta serie de elementos podría estar vacía, es decir, no contener elemento alguno.
* Una Repetición (o iteración) consiste en un único elemento que se repite una y otra vez, o ninguna, hasta que cierta condición sea satisfecha. O lo que es lo mismo, de 0 a n veces, mientras se satisfaga una cierta condición.
* Una alternativa (o selección) consiste en dos elementos, de los que sólo ocurre uno u otro, dependiendo de cierta condición. Aunque en principio se definen sólo dos elementos para definir la selección, no hay ningún problema en permitir más de dos ramas, siempre que sólo ocurra una de ellas, dependiendo siempre de dicha condición (en realidad es el resultado de des-anidar varias selecciones anidadas).

No obstante[[21]](#footnote-21) su importancia para la ciencia de la computación, el artículo de Bohm y Jacopini fue poco menos que ignorado, ya que ellos se limitaron a demostrar matemáticamente que esto era así (lo que hoy en día se conoce como Teorema de Bohm y Jacopini), pero no rompieron lanza alguna para defender que programar en base al uso de las tres estructuras fuera bueno y recomendable. Y todo el mundo siguió igual. Hasta que uno de los científicos de la computación más reputados de entonces, y de todos los tiempos, el holandés Edsger W. Dijkstra publicó en 1968 su famoso artículo "Go To Statement Considered Harmful", en el que cargó con armas y bagajes contra el uso (y el abuso) de la sentencia de bifurcación, abogando fuertemente por el uso exclusivo de las tres sentencias básicas.

La (reducida) comunidad informática de entonces aceptó bien, en general, la recomendación del maestro, pero seguía habiendo dificultades: todo esto está muy bien, pero, ante una necesidad determinada, cuando hay que resolver un problema concreto, ¿cómo diseñamos y codificamos el programa usando exclusivamente las tres estructuras?[[22]](#footnote-22) En definitiva, ¿cuál debe ser el método a seguir? Por tanto, no fue hasta que se comenzaron a formalizar métodos de Diseño Estructurado de Programas que se comenzó a considerar en serio la posibilidad de utilizar la técnica en el mundo real. Y el primero que realizó esta necesaria formalización fue el ingeniero francés Jean Dominique Warnier, a la sazón trabajando en Bull, que publicó en 1972 su libro "Logique de Construction de Programmes". Proporcionaba, por primera vez, un método sencillo y potente para diseñar un programa usando sólo las tres estructuras, utilizando llaves (estas llaves: "{"[[23]](#footnote-23)) que permitían definir el nivel siguiente, que a su vez se descomponía en otros, usando más llaves[[24]](#footnote-24)...

Se fijaba en los datos que tenía que tratar el programa para obtener la estructura correcta de tratamiento. Y obligaba siempre a especificar que cada bloque lógico tuviera un principio y un fin, aunque no hubiera nada que hacer allí, y estuvieran vacíos. Esto último, por muy completo que fuese, que lo es, fue lo que menos gustó al informático de calle, como podéis suponer.

En España, Bull tradujo del francés y publicó el libro de Warnier con el nombre de "La Programación Lógica". La publicación de este método no tuvo, sin embargo, la repercusión que hubiera debido, por dos causas: primero, no se centraba en ningún lenguaje concreto, es decir, no daba instrucciones concretas sobre cómo llevar a la práctica sus predicados en Fortran, o Cobol, Assembler o lo que fuera. Y segunda, ¡era una cosa "not invented here"! Como era obra de un francés, y publicada originalmente en francés, el mundo anglosajón no entró mucho al trapo... Entonces, igual que ahora, la informática se escribía en inglés, en inglés americano, para más señas. Y Jean Dominique Warnier era francés, había cometido la tropelía de publicar originalmente en francés... y para colmo, trabajaba en Bull, una empresa francesa.

En 1974 Marie Thérèse Bertini publicó, junto a Yves Tabourier como coautor, "Le Cobol Structuré" (en francés, claro está). Básicamente el "método Bertini" derivaba del de Warnier. Lo importante fueron sus normas para codificar la cosa resultante en Cobol (y sólo en Cobol; no publicó nada, que yo sepa, para ningún otro lenguaje). No eran muy complicadas, y básicamente se centraban en tres aspectos:

* Por un lado, la obligatoriedad de codificar los inicios y finales de cada bloque (mediante un PERFORM INICIO-TAL-COSA y PERFORM FIN-TAL-COSA, respectivamente), incluso aunque alguno estuviera vacío (porque no hubiera nada que hacer allí), en cuyo caso llevarían sólo la instrucción EXIT, que no es tal instrucción, como imaginaréis, pues sólo sirve para que un párrafo vacío no esté... vacío.
* Por otro, la prohibición de codificar PERFORM .. THRU ... Esto forzaba a que para codificar una alternativa, hubiera que hacerlo así: IF tal y tal PERFORM FULANO ELSE PERFORM MENGANO. Por tanto, generaba una gran profusión de procesos anidados en cascada.
* Y por fin, la prohibición total y absoluta para siempre jamás de usar el GOTO.

El método tuvo un cierto éxito, y se enseñó bastante en centros de formación durante unos años. Pero no era perfecto. Bien está que se prohíban los GOTOs indiscriminados, pero es que, si quieres usar toda la potencia del Cobol, en ciertos casos es obligatorio usar GOTOs. Por ejemplo, si codificas con más de una sección, lo que es bueno y recomendable por claridad, necesitas en todas ellas menos la principal dirigirte al final físico de la sección para poder devolver el control al proceso llamador. Y no se me ocurre otra forma de hacerlo que con un GOTO. Es decir, usar Bertini implicaba codificar todo el programa en una sola sección, y cuando tienes dos o tres mil líneas de código en la PROCEDURE DIVISION... no es exactamente una buena idea.

Hasta que, en 1975, el científico de la computación inglés Michael A. Jackson publicó su "Principles of Program Design". Y el tema se solucionó. Literalmente. Definitivamente. Ya no se volvió a publicar ningún otro método general de Diseño Estructurado de Programas, pues no había mucho más que añadir. Resolvió tanto el procedimiento formal de diseño, dotándole de una lógica aplastante, como esos "casos especiales" que molestaban a los otros métodos de diseño estructurado. Y además, preconizaba una forma de codificar en los lenguajes más importantes de la época, significativamente Cobol, Fortran y PL/1. Lo más impactante del método es que todo el diseño de la estructura del programa se basa en lo único que es completamente conocido, fijo y seguro: los datos. Efectivamente, todo programa que se precie (y que sirva para algo) debe leer y escribir ciertos datos, que estarán en cualquier tipo de soporte, y que tendrán una estructura determinada, conocida y fija (si no es así, malamente vamos a poder programar nada). Y a partir de la estructura intrínseca de los datos, va siguiendo unos pasos concretos y muy bien definidos que terminan con el programa codificado en pseudocódigo, o directamente en el lenguaje que sea. La idea es que, dado un problema determinado que se aplica sobre ciertos datos conocidos, sólo haya una solución posible. Veamos sucintamente el método, que preconiza cinco pasos secuenciales:

1. Establecer la Estructura de los Datos. De todos los datos, por separado, tanto los de entrada como los de salida.
2. Establecer las Correspondencias entre Datos. Para que haya correspondencia entre dos entidades de dos conjuntos de datos, sean de entrada o de salida, debe darse que: (a) ambas tengan el mismo número de elementos, y (b) éstos estén en el mismo orden en ambas entidades, o sea, ordenados de la misma forma.
3. Crear la Estructura del Programa. Para ello, cada entidad con correspondencias da origen a una única entidad, del mismo tipo, en la estructura final del programa, y el resto se insertarán en su lugar correspondiente, sin cambiarles de naturaleza. Como en este paso tratamos ya con procesos y no con datos, el propio nombre de las entidades cambia. La entidad de datos "CUENTA", por ejemplo, pasará a llamarse ahora "PROCESAR CUENTA".
4. Listar y asignar las operaciones individuales. Jackson propone un checklist de operaciones, que debe cumplimentarse en orden, para que no se olvide nada:  
    a) Instrucciones de finalización del programa.  
    b) Instrucciones de cierre de ficheros.  
    c) Instrucciones de apertura de ficheros.  
    d) Instrucciones de escritura en ficheros.  
    e) Instrucciones de lectura de Ficheros.  
    f) Instrucciones de movimiento de campos, entre ellos los de claves de ruptura.  
    g) Instrucciones de cálculo.  
   Una vez obtenida la lista, se asignan por riguroso orden las instrucciones a la estructura. ¿Dónde? Siempre en secuencia, y en aquél lugar donde la instrucción se ejecute una y sólo una vez, teniendo en cuenta las operaciones ya asignadas, para situarla antes o después de ellas. Siguiendo el orden preconizado en el checklist, es fácil determinar el lugar exacto donde asignar cada instrucción.
5. Generar el pseudocódigo. Muy fashion en esa época, pero nadie escribía ya el pseudocódigo, sino directamente el programa. Tiempo que se ahorra uno... Para ello, Jackson propuso un sistema de codificación de sus estructuras en los lenguajes más usados del momento, sobre todo Cobol, Fortran y PL/1.

¡Hala! Ya tenemos el programa escrito. Un auténtico chollo. Sencillo, rápido, bien definido... Además, catorce programadores diferentes a cuál más raro, obtendrán la misma estructura, puesto que los datos son como son, están perfectamente definidos, y todo lo demás se deriva de ellos... Más o menos... O así...

Bien, terminado el paseo por el Método de Jackson de Estructuración de Programas, desde entonces prácticamente no se ha avanzado más en la teoría, porque todo había quedado establecido. Jackson hizo un trabajo formidable, y realmente no dejó mucho más que rascar. Y fue entonces, principios de los ochenta, cuando nos pusimos en España a utilizar técnicas de Programación Estructurada en empresas e instituciones. Pero antes de eso, hubo que dar a conocer la nueva técnica a los responsables de Informática, para convencerlos de que era mejor para sus negocios adoptarla. Como podéis imaginar, en aquellos años los responsables de Proceso de Datos no estaban demasiado al tanto de las novedades que se producían en la profesión, primero por su origen (casi ninguno era informático de profesión, sino que provenían de puestos administrativos, y su formación era en otras áreas, quien la tenía, porque, por ejemplo, uno de los Directores de Informática que yo conocí comenzó su carrera en el Banco como botones, lo que dice mucho en favor de su capacidad), y luego, por el dichoso inglés. Apenas se hacían salidas a Conferencias el Extranjero, y no se leían muchas revistas especializadas americanas o inglesas, que además de ser incomprensibles (por el idioma) eran caras... pero caras.

Hicimos nuestros pequeños pinitos, sí, pero no conseguíamos convencer a nuestros mayores de que fuera conveniente la adopción generalizada de la técnica, entre otras cosas, porque éramos simples programadores... o sea, unos mindundis. Y de pronto, algo ocurría: ante la necesidad que acontecía de vez en cuando de escribir un programa complicadísimo de resolver (de esos a los que nadie se atrevía a hincarle el diente), entonces te lo encargaban para que lo escribieras de forma estructurada, con la coletilla de "Hala (listillo: esto lo pensaban pero no lo decían), a ver si eres capaz de resolver este embolado con esa cosa rara nueva...", y todo esto con una aviesa sonrisilla de diablillo que te llegaba hasta el hígado... Y entonces el avispado programador, tocado en su amor propio, se ponía a diseñar y codificar el maldito programa... y resulta que, sorprendentemente, en mucho menos tiempo del que se suponía, el programa estaba listo, y encima funcionaba bien rápidamente (no, a la primera, no, que a la primera no funciona nunca nada...), y además no se eternizaba en máquina... Y entonces era cuando las fuerzas vivas comenzaban a darse cuenta de la potencia del asunto, y a tomárselo en serio.

Por lo que yo sé, procesos similares fueron ocurriendo poco a poco en diferentes Centros de Cálculo, conforme las historias de éxito iban siendo conocidas, y además, aunque las entidades (los Bancos, por ejemplo) fueran competencia unas de otras, entre los informáticos de entonces había una sana camaradería, por lo que unos íbamos contando a los otros nuestras experiencias, ayudando a desechar lo que no había funcionado y a extender lo que sí. Así que, a lo largo de la primera mitad de los ochenta, las empresas se fueron convenciendo gradualmente de la bondad de la Programación Estructurada, y sobre 1983-84 ya todos los Centros de Proceso de Datos importantes habían adoptado la técnica.

# 1986

La informática "tradicional", es decir, la gran informática empresarial o gubernamental, sobre todo la de gestión, en una palabra, la de los mainframes, había llegado a un punto de desarrollo importante. Todas las grandes empresas, del sector que fueran, ministerios, etc., tenían ya su mainframe funcionando, y se estaban escribiendo muchas aplicaciones nuevas, bien para sustituir aplicaciones que habían ya quedado obsoletas, bien para solucionar nuevas necesidades, sobre todo, derivadas de los emergentes procesos online. Es decir, las grandes empresas tenían ya su mainframe. Varias marcas ofertaban grandes ordenadores: NCR, BULL, Siemens, Control Data, UNIVAC, BURROUGHS...Estas dos últimas se fusionaron en 1986, para dar lugar a UNISYS. Aunque todas estas marcas ofrecían ordenadores de muy buena calidad, y vendieron bastantes sistemas en las dos décadas anteriores, en los ochenta la carrera del mainframe ya tenía un claro ganador: IBM.

La serie 303x, sustituta del mítico 370, había tenido ya un gran éxito, pero el lanzamiento de la serie IBM 308x literalmente arrasó a la competencia, que poco a poco dejó el campo libre. De hecho, a partir de la segunda mitad de los 80, la principal competencia de IBM en mainframes fueron, curiosamente, máquinas de otros fabricantes (Fujitsu, Amdahl, Hitachi...) que, producto de la ingeniería inversa, eran totalmente compatibles con las de IBM (y más baratas, desde luego), y funcionaban con su mismo software. Eran los clónicos del momento, conocidos como PCM (Plug Compatible Machines), y tuvieron su momento de gloria a raíz de una sentencia del Tribunal Antitrust de EEUU que obligó a IBM a no negarse a vender su software a los clientes que tuvieran este tipo de máquinas. Además, empezaron a venderse dispositivos periféricos (discos, cintas magnéticas, impresoras...) compatibles también con los de IBM, de otras marcas; para IBM, evitar estas ventas de periféricos era mucho más difícil de frenar que las de las propias CPU's. IBM reaccionó también, incluyendo cada vez más microcódigo en sus ordenadores, de tal manera que fuera más difícil para la competencia descifrar su funcionamiento mediante ingeniería inversa... pero esta estrategia, en realidad, le hizo ganar apenas unos meses de tranquilidad. En cualquier caso, los mainframes de IBM eran las máquinas más vendidas, y no digamos ya su software, que había alcanzado una posición de virtual monopolio en este mercado.

Pero IBM tenía un problema con esto: No hay tantas grandes empresas o instituciones en el mundo. La mayoría tenían ya, o tendrían en unos pocos años, un gran mainframe suyo (o quizá un PCM con software de IBM), que habría que ir ampliando con el tiempo, irlo sustituyendo por otro más moderno en su debido momento, y pagar religiosamente cada año el alquiler del software. Todo esto reportaba pingües ingresos a IBM (de hecho lo sigue haciendo: la División de Mainframes sigue siendo la más rentable de IBM en la actualidad), pero no bastaba. IBM quería más. De hecho, siguiendo la más estricta lógica empresarial, IBM lo quería todo. Como veis, cuando se acusa a según qué Compañía de Software actual de la misma cosa... no hay nada nuevo bajo el sol.

* En el mercado de los pequeños ordenadores, los "minis", adecuados a pequeñas empresas que no necesitaban tanta potencia como para poder amortizar un mainframe, en entornos ingenieriles, etc, IBM tenía también líneas de productos específicos: Serie/1, System/3 y sus múltiples sucesores en el tiempo: System/32, System/34, System/36 y System/38, que dieron lugar al final, en 1988 al AS400 (desde hace pocos años, rebautizado como Serie i, o mejor, iSeries). Todas estas máquinas son el reino del RPG (acrónimo de "Report Program Generator"), y luego de su sucesor, el RPG II; también tienen otros compiladores, Cobol desde hace muchos años, así como C, etc. Su arquitectura actual los hace tan robustos y seguros como los mainframes, más pequeños y menos potentes, menos complicados de administrar y más baratos; siguen siendo sistemas con un importante mercado, sobre todo en las empresas medianas.  
  Hasta 1990 no tuvo IBM una gama de ordenadores basados en UNIX: los RS/6000 (por RISC Systems, ya que los procesadores que utilizaron eran de tecnología RISC: el "IBM Power", primer procesador RISC utilizado comercialmente de forma masiva), cuyo Sistema Operativo es AIX, la versión propietaria de IBM de UNIX (del mismo modo que Solaris es la de Sun, o HP/UX la de Hewlett-Packard: ya sabéis: hay diez o quince UNIX todos iguales... y todos distintos); la gama de Sistemas UNIX (bueno, AIX), los RS/6000, continúa siendo parte fundamental de la oferta por parte de IBM en la actualidad. Eran todos de tecnología propia (chips propios y específicos, dispositivos y periféricos específicos también de IBM, software propio e incluso lenguaje de programación propio: el RPG).  
  Es decir, seguían la misma técnica que había utilizado con éxito la División de mainframes: usar exclusivamente tecnología propia bajo patente de IBM, sacando el máximo partido a los excelentes Centros Tecnológicos que IBM tenía repartidos por todo el mundo (incluida España). Pero igual que sus mainframes habían triunfado definitivamente, en el área de las máquinas pequeñas IBM no triunfó en absoluto. Vendieron muchos sistemas, desde luego, pero aquí la competencia era feroz por parte de muchos otros fabricantes de hardware y software que disponían también de excelentes productos: DEC (Digital Equipment Corporation), Data General, Olivetti, Nixdorf, Philips, Siemens, Hewlett-Packard, Bull, etc). Cubrían una amplia gama de necesidades, con sistemas operativos bien propios (como el VMS del VAX de Digital) o bien usaban alguna versión del UNIX de la época. Entre tanto fabricante, IBM era únicamente un competidor más, un lobo más en la jauría, y seguramente ni siquiera el más exitoso en este segmento de mercado (seguramente, aunque quizá me equivoque, lo fuera Digital en aquellos años).
* En cuanto a los terminales especializados (es decir, el terminal financiero, el terminal punto de venta, etc), IBM tenía (cómo no) oferta en todos estos campos y (cómo no) con tecnología propia: el Sistema Financiero IBM 3600; para comercios, el Sistema de Terminal Punto de Venta IBM 3650, y para otras áreas de negocio tenía también una serie de equipos especializados que no voy a detallar, pues sería tedioso: IBM estaba presente en casi todas las áreas de negocio habidas y por haber. No sólo toda la tecnología que IBM ofrecía era propia, sino que era casi incompatible entre sí: no había prácticamente ninguna transferencia entre la tecnología de los mainframes y la de los terminales financieros, o la del Sistema/34, por ejemplo (cada sistema se desarrollaba en un laboratorios diferente, y, por lo visto, debían estar todos peleados entre sí, aunque en IBM siempre dijeron lo contrario... ¡faltaría más!). También en estas áreas IBM afrontaba una dura competencia de otros fabricantes, que de hecho vendían mucho y muy bien, incluso para conectarse a los inevitables ordenadores centrales de IBM. Philips, Nixdorf y Olivetti, por ejemplo, en el mercado europeo sobre todo, tenían excelentes Sistemas de Terminal Punto de Venta para tiendas minoristas, y Sistemas Financieros, tan eficaces como los propios de IBM, y más baratos. En una palabra, tampoco en este segmento especializado IBM llevaba la voz cantante.

Así las cosas, estaba comenzando a aparecer un nuevo segmento de mercado, hasta hacía muy poco tiempo inexistente: El del Ordenador Doméstico. El pionero en este campo fue sin duda Apple, con su Apple I de 1976 (fabricado a mano sobre pedido, en total unas 200 unidades, y que se entregaba en forma de kit, con una placa base, la CPU, la memoria y un chip de video, de sólo texto, para que el usuario lo montase), y sobre todo con su Apple II, lanzado en 1977, que se fabricó ya de forma industrial, y que se vendía ya completo, con su carcasa, su teclado y todo. Eran máquinas muy poco potentes, con unos pocos Kb de RAM, lector/grabador de cinta magnética, y sin disco (aunque pronto se escribió el driver para manejar un disco flexible en el Apple II). En cuanto a pantalla, aunque tenían la suya propia, eran también capaces de conectarse a un aparato de televisión. Pero eran baratas, no con criterios actuales, claro, pero sí comparadas con cualquier alternativa existente en la época: el precio de la configuración más cara (la que llevaba nada menos que 48KB) fue inicialmente de alrededor de 2.600 dólares. Estos incipientes ordenadores domésticos se usaban, sobre todo, no nos engañemos, para jugar. La escasísima capacidad de almacenamiento, la dificultad en recuperar y guardar datos externos, y, por qué no decirlo, su poca fiabilidad, lo hacían poco menos que inviable para cualquier aplicación profesional.

Otros fabricantes como Texas Instruments, Commodore, Atari, etc, siguieron sus pasos, y construyeron sus propios ordenadores domésticos, cada vez más completos y más baratos.

Y mientras tanto... ¿qué hacía el líder absoluto?, os preguntaréis, ¿qué hacía en este campo IBM? si es que hacía algo... Desde luego, en IBM no tenían un pelo de tontos, así que ya en 1973 o 1974 se dieron cuenta de lo prometedor que para su cuenta de resultados podría ser esta nueva gama de ordenadores. Y tomaron la decisión de ir tomando posiciones en este tipo de tecnología. ¿Cómo hacerlo? Pues... ni más ni menos que como siempre había hecho las cosas IBM. Así que, aplicando una vez más su mantra empresarial, es decir, vender sólo tecnología propia, comenzaron el diseño de un ordenador personal (bueno, no sé si exactamente personal, pero, al menos, pequeño): el ordenador portátil IBM 5100. Lanzado en 1975, fue una maravilla de diseño y empaquetamiento: dotado de un lector de cinta magnética (en casete), teclado y pantalla, pesaba sólo 25 Kilos de nada, y costaba únicamente 9.000 dólares de nada el modelo de 16 Kb, y 20.000, el de 64 Kb. Para que os hagáis una idea, el Apple II, un par de años más tarde, tenía un precio de 1.298 dólares con 4Kb, y 2.638 dólares si llegaba a 48Kb, como ya comenté hace un momento. O sea, que el 5100 era... un auténtico chollo. Efectivamente, fue una maravilla de diseño... y un sonado fracaso comercial.

Aunque su venta continuó hasta 1981, nunca se vendieron muchas unidades de este IBM 5100. Así que, hacia 1980, el mercado de los ordenadores domésticos (pequeño todavía) estaba claramente dominado por Apple, con otros competidores como Commodore, con su Commodore-64, etc, entre los que en la práctica no estaba IBM. Y las pocas máquinas que se vendían las compraban sobre todo aficionados e ingenieros, para chapucear sus programillas en Basic o, fundamentalmente, para jugar a los primitivos juegos de ordenador de entonces. En realidad, se vendían pocas máquinas, si lo miramos desde una perspectiva actual, pero eran desde luego muchas si lo comparamos con el resto del mercado de ordenadores en la época.

Entonces, de repente, en 1979 todo cambió. Apple lanzó la Hoja de Cálculo Visicalc como un producto más para su Apple II... y entonces, de golpe y porrazo, se materializaron de la nada millones de potenciales usuarios que antes ni sabían ni querían saber nada de estos "ordenadores personales": Administradores, Contables, Gestores, Financieros, etc., hallaron una poderosa herramienta para ayudarles en sus labores profesionales. Y gracias a Visicalc, las ventas de Apple se dispararon. Visicalc fue al poco tiempo portada a otros ordenadores personales de la época, y comenzaron también a aparecer otras hojas electrónicas clónicas, aprovechando el hecho de que a sus autores, Dan Bricklin y Bob Frankston, no se les ocurrió patentar el invento... y así nació la informática moderna[[25]](#footnote-25).

A IBM le pilló esta vez, cosa rara, a pie cambiado. No habían anticipado este giro copernicano del mercado. Tenían, sí, un ordenador pequeño (no sé si llamarlo doméstico, personal, o quizá portátil) propio susceptible de ser ofertado, pero comercialmente muerto. Y rápidamente se dieron cuenta de la potencialidad de estas nuevas máquinas para ofrecérselas a sus clientes de toda la vida: las empresas. Pero... no tenían ningún producto que ofrecer a sus clientes. Dos alternativas tenía IBM: La instintiva, la que llevaban en los genes, es decir, seguir su práctica habitual y crear un nuevo sistema de tecnología propia poco menos que de la nada, o bien hacer de la necesidad virtud, y apoyarse en tecnología ya existente de otros fabricantes para construir su propio ordenador personal. Con una decisión pragmática, en vista del resultado comercial del 5100, optaron por la segunda, pues mandaron los plazos: no podían permitirse un desarrollo de años hasta tener su producto listo, así que optaron por la integración: un cambio radical en la estrategia tradicional de IBM.

Reunieron, pues, un grupo de ingenieros que, en menos de un año, habían seleccionado componentes prometedores del mercado, preferentemente de pequeñas compañías que no tendrían problema ninguno en aliarse con el gigante, y los habían ensamblado para lanzar, a mediados de 1981, el IBM 5150, más conocido como IBM PC. Entre estas pequeñas compañías estaba cierto pequeño fabricante de chips llamado Intel, y una pequeña compañía de Seattle conocida por su Basic, y por haber creado Xenix, una versión de UNIX de 16 bits: una tal Microsoft, a quien encargaron la confección del Sistema Operativo para el IBM PC. Hubiera podido IBM elegir CP/M, de Digital Research, líder indiscutible del mercado de Sistemas Operativos para este mercado, pero fueron reticentes al acuerdo (los de Digital Research, quiero decir), seguros de su superioridad tecnológica, mientras que Microsoft firmó todo (casi todo) lo que IBM le puso por delante. Y desarrolló (o compró, o adaptó, no está muy claro el asunto) PC DOS 1.0 (lo que luego se convertiría en MS/DOS). Aún así, los primeros PC's de IBM ofrecían ambos Sistemas: PC DOS y CP/M, para que los usuarios eligieran, sólo que este último, que era mejor, era también más caro... y eso le costó su supervivencia.

Lo que no ofrecía IBM era UNIX para su PC (o sea, el Xenix de Microsoft, que Santa Cruz Operations portó a los procesadores Intel en 1983, dado que Microsoft no estaba muy dispuesta a apoyar nada que no fuera su DOS... e IBM tampoco). En una palabra, quien quisiera un SCO UNIX en su PC, debería de comprarlo aparte, instalárselo, etc. Y en aquellos tiempos heroicos... fue simplemente demasiado para UNIX. Hasta bastantes años más tarde no comenzó este Sistema Operativo a tener una presencia significativa entre el parque de PC's.

Estamos en 1981, e IBM acaba de lanzar su IBM PC: Procesador Intel 8088 a 4,77 Mhz (de un solo núcleo, por si os lo estábais preguntando), 16 Kb de memoria (ampliable a 256 Kb), lector/grabador de cinta magnética o unidad de disco flexible de 5,25 pulgadas, monitor monocromo, y mayormente PC/DOS como sistema operativo. Y como software de gestión, la inevitable hoja de cálculo, Visicalc o cualquiera de sus clones, incluyendo, desde 1983 a Lotus 1-2-3, que se impuso rápidamente como líder hasta fines de los noventa. Y algún producto para escribir documentos, inicialmente el DisplayWriter (aunque antes que eso aún ofrecía cosas más exóticas, como Script, por ejemplo, que era algo como HTML, pero peor). Nosotros en España nos enteramos algún tiempo después, pues no fue hasta fines de 1982 que IBM comenzó a vender en España el IBM PC (supongo que parte del tiempo lo emplearon en traducir el PC/DOS y los manuales al español); los siguientes lanzamientos ya llegaron aquí con muy poco tiempo de demora sobre el lanzamiento en Estados Unidos. Este IBM PC no era gran cosa, incluso comparándolo con sus competidores de la época. Pero la maquinaria de marketing de IBM era formidable, y en pocos meses había convertido al IBM PC en un impresionante éxito comercial. Se vendió a particulares (jugadores, aficionados e ingenieros, los de siempre, vamos), aunque en este mercado Apple II y los demás mantuvieron sus ventas, pero sobre todo se vendió a Empresas, cuanto más grandes, mejor, donde la presencia comercial de IBM era ya muy importante: Contables, Administrativos, Gestores, Directores, Secretarias de Dirección, etc comenzaron a ser usuarios de un PC.

Dos años después, IBM sacó al mercado el IBM XT, ya con disco duro de 10 Mb, y a fines de 1984 el IBM AT, equipado con un procesador Intel 80286, disco duro de 20 Mb, y pantalla de color CGA, y provisto de un flamante MS/DOS 3.0, mucho más fiable que las versiones anteriores. Por cierto, este IBM AT, con su reluciente MS/DOS 3.0, y made in Ireland, fue el primer PC que tuve yo en mi mesa, para mí solito, a mediados de 1986. Con sus 20 Mb de disco duro... ¡no había manera de llenarlo!. El Sistema Operativo venía en dos (creo recordar) floppies de 5,25 pulgadas y 360Kb; en uno lo que era el sistema en sí, y en el otro, los programas adicionales. Era como revivir la época del NCR Century 200, pero esta vez con 640Kb de memoria RAM, así que, yo, literalmente, nadaba en la abundancia. Estas máquinas eran buenas, pero eran terríblemente caras. Y, no obstante, las empresas las compraban... ...E IBM nadaba en el éxito... ...Y sin embargo... ...Y sin embargo, no era suficiente. Vale que eran los ordenadores personales más vendidos, pero las cifras de venta, jugosísimas, seguían sin satisfacer a IBM. Veamos por qué, recordando que estamos en 1984, año arriba año abajo:

Los años ochenta vivieron una importante crisis bancaria en España: muchas entidades pequeñas y medianas fallecieron y debieron ser fusionadas con otras, o adquiridas y puestas bajo la tutela de las de mayor tamaño. Algún tiempo después, también las entidades más grandes tuvieron problemas, algunos de ellos muy serios, y se fueron produciendo fusiones entre ellas, dando origen a esos Bancos y Cajas de Ahorro tan conocidos en cuyo nombre oficial encontramos una retahíla de nombres de extintos bancos absorbidos de grado o por fuerza a lo largo de los años. La causa de estas convulsiones fue, entre otras, la mayor apertura del mercado, obligada por la necesaria incorporación de España al Mercado Común Europeo (que se materializó en 1986), que hizo que la Banca española dejara de ser exclusivamente nacional y protegida por mil leyes, para competir en un mercado amplio, europeo en principio, luego global. Algunas entidades se adaptaron, pero otras no fueron capaces, y quedaron por el camino. Consecuencia directa de este nuevo entorno competitivo fue el tremendo aumento de la oferta bancaria a la población general: muchísimos particulares de primeros de los ochenta no tenían cuenta bancaria. Aumento de oferta, tanto en productos ofrecidos, como en puntos de venta. Fueron los años de la enorme proliferación de oficinas bancarias por todo el territorio, y fueron también los años donde se comenzó la comercialización masiva de nuevos productos, el primero, la propia Cuenta Corriente, y después otros muchos, como Tarjetas de Crédito o Débito (que casi nadie tenía en España antes de esos años), Supercuentas, Fondos de Inversión, la generalización de la domiciliación de recibos,... Pero además de esta creciente apertura comercial, también se produjo una importante apertura tecnológica: Todas las operaciones, casi sin excepción, deberían ser online; todos los empleados de las sucursales deberían tener acceso al sistema, y no sólo los cajeros, como hasta entonces; se empezaron a instalar cajeros automáticos a cascoporro, etc. La tecnología en sí se convirtió en un elemento de marketing más para las entidades financieras. En una palabra, la inversión en tecnología necesaria para adaptarse a los nuevos tiempos comenzó a crecer fuertemente... y no precisamente en el entorno de los ordenadores centrales, que sí que crecía, pero no tanto, sino en el equipamiento de las oficinas. Y ordenadores centrales hay uno por entidad, quizá dos, pero oficinas... las hay por cientos, o por miles. Y es en este punto donde estas dos historias más o menos paralelas convergen: Porque IBM se encontró con dificultades en su mercado tradicional de tecnología, por la competencia de otros fabricantes que sabían bien lo que se hacían, y porque su política de "bueno-y-caro" comenzó a tener problemas para ampliar ventas en un mercado bancario en expansión que estaba requiriendo fuertes inversiones en tecnología. Básicamente, los responsables de Tecnologías de la Información de los Bancos y Cajas de Ahorro le decían a los comerciales de IBM: "Vuestra tecnología es la mejor, vale. Pero no puedo pagarla. Así que me conformaré con la de Fulanito, que, aunque es peor, sí que puedo pagarla. No es personal, son negocios...".

¿Cómo reaccionó IBM? Inventando una solución brillante... aunque también dolorosa (para ellos, quiero decir): Sacrificó su querida tecnología propia especializada y convirtió a los IBM PC's en los terminales bancarios. Y, más adelante, en casi cualquier terminal (de punto de venta, "pantalla tonta", etc). Gracias a sus especificaciones abiertas, a su capacidad de conexión mediante las primitivas Redes Locales (Token Ring, en el caso de IBM) y a la facilidad de ampliación usando los slots libres, este trabajo pudo hacerse en un espacio corto de tiempo. Para ello, diseñó placas específicas que permitían a un PC emular el comportamiento de los terminales bancarios especializados, conectados a los dispositivos extraños que se necesitaban (impresoras bancarias, lectores de cheques, dispensadores de dinero, etc) y con la misma conectividad con los sistemas centrales que los terminales especializados. Y escribió también el software de emulación que permitiera correr el mismo software y las mismas aplicaciones que ya funcionaban en sus IBM 3600 ó 4700 (protección de la inversión), o construir el software de las oficinas (para los nuevos clientes). Con ello consiguió, fundamentalmente, dar una respuesta moderna y eficaz a las necesidades de tecnología de los Directores de TI y, sobre todo, a un precio más que competitivo. Como, por aquella época, también empezaron a aparecer tarjetas de emulación 3270 (que eran capaces de hacer que el PC emulara a un terminal "tonto" del mainframe), también se comenzaron a instalar PC's en lugar de pantallas tontas en los centros administrativos, pues además de su uso primario (acceder a las aplicaciones online del mainframe), también servirían como terminal ofimático (con las muy limitadas posibilidades de la época: hoja electrónica y procesador de textos, pero esto también iba a cambiar en unos pocos años). El éxito fue importante, nuevamente. Comenzaron a proliferar PC's en oficinas bancarias, tiendas, comercios, y otros establecimientos donde poco antes era impensable que estuvieran, así como en la mayoría de oficinas centrales, y muchísimos de ellos, de IBM, generando los años dorados (o sea, de oro) para el coloso del momento. Para que os hagáis una idea de los dorados que eran, a mediados de la década de los ochenta, IBM facturaba ella sola en equipamiento informático más que los siete siguientes competidores (DEC, Univac, NCR, Burroughs, Hewlett-Packard, etc) juntos. Y fueron felices y comieron perdices... Pues no. La competencia tampoco se quedaba atrás, así que reaccionaron. Vaya si reaccionaron...

# 1988[[26]](#footnote-26)

# 1989

La aparición de la Bases de Datos Relacionales había significado un shock para la profesión: implicaban la utilización de técnicas de diseño de Base de Datos bastante diferentes de lo que estábamos acostumbrados, mucho menos "personalistas", si se me permite la expresión, y mucho más industriales. Es decir, comenzaba a vislumbrarse el mismo camino que, pocos años antes, habían seguido las técnicas de programación, con la aparición y rápida adopción de la Programación Estructurada. Por tanto, el sentimiento de los profesionales de la época (mediados de los ochenta), y entre ellos el mío propio, era algo así como: "Vaya, por fin comienza a ser posible unificar formas y métodos, y encontrar un lenguaje común que nos permita una comunicación precisa entre nosotros...". ¡Cuán equivocados estábamos...! Pero, como entonces no lo sabíamos, hicimos grandes esfuerzos para conseguir ese "idioma común". Porque la situación estaba cambiando a pasos agigantados. Los últimos años de los ochenta y primeros de los noventa fueron de un enorme crecimiento en todos los aspectos de la informática: Crecimiento en potencia y variedad de las máquinas, en Sistemas instalados, en Aplicaciones funcionando (ya, casi todas, online) y, sobre todo, de fuerte crecimiento del personal de los Departamentos de Informática (o Proceso de Datos, como seguían llamándose mayoritariamente por entonces: no había llegado aún la moda del marketing a los nombres de los Departamentos dedicados a esto). Casi podría asegurar que en el plazo de tres o cuatro años, el plantel de informáticos al menos se duplicó en la mayoría de grandes empresas, y en algunas, bastante más. Y no fue nada fácil encontrar a tanto informático. Doy fe. ¡La de currícula que me leí, y la de entrevistas que pude hacer esos años! Costó un gran esfuerzo poder incorporar tanto técnico informático a las empresas, simplemente porque en el mercado no los había.

Todas las Universidades Españolas quizá podían graduar a, digamos, mil informáticos anuales, seguramente bastantes menos... totalmente insuficiente para tales crecimientos, por lo que no hubo más remedio que echar las redes en los caladeros de cualquier otra especialidad, sobre todo las técnicas (lo que por entonces se conocía como "Carreras de Ciencias"): Matemáticas, Física, Biología, Geología, Ingenierías, sobre todo Teleco (aunque menos), ¡incluso Medicina!

Eran éstas Carreras que proporcionaban una buena preparación de base, algún conocimiento de informática, y que, con la excepción de Medicina, no tenían mucha salida en España. Así que las grandes empresas nos lanzamos a contratar Licenciados, Diplomados, Peritos o lo que pilláramos (sí, también algún que otro recomendado, por si lo os lo estáis preguntando, pero pocos), y a formarlos en la tecnología pertinente, mayormente Cobol, DB2, IMS o CICS, según el caso, y toda la parafernalia mainframera asociada: ISPF, JCL, SPUFI, etc. Y Programación Estructurada, naturalmente... Porque en esa época (mediados y finales de los ochenta), estas cosas mundanas ya casi no se explicaban en las Universidades españolas, a pesar de que eran las que daban trabajo a quizás siete de cada diez informáticos de los de entonces.

Pues, veréis, porque con tanta incorporación de personal comenzaba a ser evidente que era necesario llegar bastante más allá de una simple recomendación de uso de técnicas: había que asegurarse, en lo posible, de que tanto el procedimiento de Diseño de las Aplicaciones como la Documentación asociada estuvieran formalizados, uniformizados, reglamentados, constreñidos, de tal modo que cualquier persona pudiera leer, comprender y, llegado el caso, modificar cualquier Aplicación sin necesidad de apelar a la memoria o el conocimiento de su creador... Esta era una situación que se daba con alguna frecuencia, y que las empresas no estaban dispuestas a que volviera a ocurrir en el futuro. Aunque igual sabéis que, en realidad, por mucho que documentes, y que mantengas la documentación, sigue siendo mucho más eficaz que realice cualquier cambio a la Aplicación alguien que la conozca, que alguien que no sepa de qué va, por mucha documentación electrónica o en papel que haya...

Por ello, parecía evidente que había que implantar un procedimiento de Diseño, Construcción y Documentación que permitiera eliminar a los "Hombres-Aplicación" (o "Mujeres-Aplicación", que también había). Y para ello, se necesitaba:

1. en primer lugar, un Método que dijera qué había que hacer, y qué no;
2. en segundo lugar, unas Técnicas de uso común para realizar Análisis, Diseño, y Programación (aunque esta última ya estaba bastante formalizada);
3. y en tercero, a ser posible, un Sistema de algún tipo que se asegure que las Normas del Método se cumplen y las Técnicas se aplican.

Estas necesidades básicas son solucionadas, como habéis intuido, por las Metodologías de Desarrollo de Software, por las Técnicas de Análisis y Diseño, y por las Herramientas CASE, respectivamente. Pero... ¿cuál era el estado de cada una de ellas a mediados de los ochenta?:

No había Metodologías de Desarrollo, tal como las conocemos ahora, en la práctica. Sí que había algunas, de las que hablaré brevemente en un momento, que eran más bien un listado de tareas y documentos que había que cumplimentar, firmar y guardar. Y esto porque, en su mayoría, fueron promovidas por ciertos Gobiernos para asegurar el cumplimiento de las especificaciones y, sobre todo, de los costes de los grandes proyectos contratados por la Administración.

En cuanto a Técnicas, hemos visto cómo la Programación Estructurada había triunfado ya, pero apenas había nada sobre cómo realizar el Diseño Técnico, ni mucho menos el Análisis Funcional. Para el Diseño Técnico, ya se escribían y guardaban con cierto formato los Cuadernos de Carga, pero a la hora de explicar lo que debía hacer el programa, se explicaba... en cristiano... como se le ocurría al Analista Orgánico. Sin ir más lejos: "Abrimos el fichero de entrada; leemos registros, y consultamos en la Base de Datos BDCUENTA si la cuenta existe; si existe, patatín, patatán... si no existe, patatán, patatín..." y así todo. Y en cuanto al Funcional, se daban reglas, sugerencias... pero el Analista, en definitiva, escribía su Análisis Funcional explicando en román paladino lo que debía hacer la Aplicación como buenamente podía, sabía y se le ocurría. Luego, en cuanto al diseño de las Bases de Datos, tampoco había mucho más que la experiencia del Analista, y las Reglas de Normalización de Codd, siempre que fueran las Bases de Datos fueran Relacionales, que si eran jerárquicas, ni eso.

Y por fin, en cuanto a las Herramientas informáticas, alguna había, pero muy orientadas a facilitar el trabajo del programador (que era, en verdad, el único que trabajaba con herramientas informáticas, el TSO, el ISPF, etc, pues el resto de roles del Desarrollo de Aplicaciones trabajaban mayormente en aquellos años con un lápiz y un papel... y gastar saliva, sobre todo, mucha saliva).

Quizá conviene resaltar que, ya entonces, el Mantenimiento de Aplicaciones era un serio problema... y tenía toda la pinta de convertirse en El Problema: la Aplicación, una vez puesta en marcha, comienza una fase de mantenimiento que supondrá seguramente, a lo largo de toda la Vida de la Aplicación, entre un 75% y un 90% del coste total dedicado a dicha Aplicación a lo largo de su vida útil. Además, las proyecciones eran palmarias: de seguir así, en unos pocos años el 90% de la plantilla de Desarrollo, o más, estaría dedicada exclusivamente al Mantenimiento... y entonces, ¿quién escribiría las nuevas Aplicaciones? Nuevo personal, claro está. Que habría que fichar o subcontratar, obviamente. Y que, una vez terminada su Aplicación, quedaría, en un 90%, dedicado a su mantenimiento. Y habría que fichar entonces más personal nuevo para escribir las nuevas aplicaciones... No parecía que el círculo vicioso originado por esta forma de trabajar resultara muy eficiente a la larga. Así que las connotaciones son evidentes: No hay que escatimar esfuerzos durante la etapa inicial de Diseño y Construcción de la Aplicación para reducir todo lo posible el coste del Mantenimiento posterior. Es decir, incrementar un 20%, un 30% ó incluso un 50% el coste del Desarrollo de la Aplicación no supondrá, a la larga, un porcentaje importante del coste total de la Aplicación (quizá un 5%-10% del total), mientras que reducir en un 20% ó un 30% el coste del Mantenimiento sí que tendrá un efecto importante en los costes finales acumulados.

¿Qué es, en realidad, una Metodología? Una serie ordenada de tareas y procedimientos, junto con las técnicas asociadas, que permiten a los profesionales del Desarrollo cumplir con su trabajo (escribir software y luego implantarlo y mantenerlo funcionando, mayormente) con la máxima calidad y eficiencia, en el menor tiempo posible, y siempre con el menor coste posible. ¿Sí? Y... ¿para qué sirve una Herramienta CASE? Para ayudar a realizar estas tareas y procedimientos con un soporte informático que nos ayude, por un lado, a seguir rigurosamente el Método seleccionado, y por otra, a asegurar en todo caso y tiempo la corrección formal de nuestro trabajo, la salvaguarda de la información, su mantenibilidad en el tiempo... Pero... ¿Con qué objetivo nos pagan nuestros generosos salarios nuestros patronos, cuál es el beneficio que los accionistas de nuestra empresa esperan obtener a cambio de nuestro trabajo, y de nuestro sueldo? **¿Cuál es, en realidad, el trabajo de nosotros, los informáticos?** ...Porque no es para "Escribir Programas brillantes", no. No es tampoco para "Diseñar Aplicaciones Chulas", ni mucho menos, como ya sabéis, ni para "Utilizar las Herramientas más novedosas", en absoluto. **A nosotros nos pagan para proporcionar a nuestros Usuarios (el resto de la empresa) una serie ordenada de tareas y procedimientos, apoyados por un soporte de software informático, que les ayude a, por un lado, seguir rigurosamente los procedimientos de trabajo, y por otro, asegurar la corrección formal de la información crítica de la empresa, su salvaguarda, la capacidad de recuperación ante errores, su mantenibilidad en el tiempo**. ¡Exacto! Con el advenimiento de las Metodologías y de las Herramientas CASE, lo que estaba ocurriendo era, de facto, que estábamos informatizando al Departamento de Informática. Ni más ni menos. ¡Sólo que no lo sabíamos! O no queríamos saberlo, quizá. A mí siempre me ha parecido muy curiosa esta esquizofrenia colectiva de nosotros, los informáticos: nos hemos opuesto sistemáticamente a que alguien nos mecanice nuestro trabajo, cuando eso es precisamente lo que hacemos nosotros continuamente... o quizá es, precisamente, por eso mismo. Es decir, los pasos lógicos que seguimos cuando nos acercamos a un Usuario cualquiera (digamos, el Departamento de Riesgos, o el de Contabilidad...) para poner en marcha una Aplicación para él, siempre han sido, más o menos:

1. **Estudio del Sistema Actual:** En él nos informamos de cómo trabaja ese Departamento, qué datos maneja y qué procesos realiza, qué interfaces tiene con el resto de Departamentos, y, en definitiva, cuáles son sus necesidades.
2. **Diseño del Sistema Propuesto:** En base a la información obtenida anteriormente, realizamos el Modelo Tecnológico, el Modelo de Datos que soportará su negocio, el Modelo de Procesos, identificando oportunidades de mejora, etc., etc. Se seleccionan las herramientas hardware y software que se necesitarán para dar soporte al futuro Sistema, se establecen tareas, plazos...
3. **Realización del Sistema:** Una vez aprobado lo que se quiere hacer, pues se hace (incumpliendo los plazos, como es tradición), se implanta (con problemas, como siempre), se forma al usuario en su uso, se da soporte, y luego se mantiene por los siglos de los siglos, amén.

Por consiguiente, si lo que deseamos es mecanizar al Departamento de Informática, lo lógico es aplicarnos la misma medicina que nosotros aplicamos a nuestros usuarios: Estudio del Sistema Actual, Diseño del Sistema Propuesto y Realización del Sistema, ¿no os parece? Pues no. No conozco ni un solo caso en España (y supongo que tampoco habrá muchos por ahí fuera, si es que hay alguno) en que esto se hiciera así, es decir, con amplitud de miras, sabiendo cuál es el objetivo final, y tomando las decisiones pertinentes. La implantación de Metodologías, Herramientas y Técnicas en Desarrollo, y en todo Proceso de Datos, se fue haciendo "a trocitos", de forma parcial, y en muchos casos forzado por la presión mediática y de las consultoras: llegó un momento, a fines de los ochenta, en que "si no ponías un CASE en tu vida, eras un neanderthal", así que se adquirieron herramientas, se adoptaron métodos, se enseñaron técnicas... pero sin un Programa Director que nos indicara de dónde veníamos, a dónde íbamos, ni, desde luego, cuáles eran los objetivos finales a obtener, cuál era la luz al final del túnel. No hubo, en definitiva, metodología alguna que usáramos para la selección de nuestra metodología, ni de nuestras magníficas Herramientas CASE... y el resultado es que, diez, doce o quince años después de aquéllos momentos cruciales en nuestra profesión, no queda prácticamente nada. Apenas hay alguna gran empresa española que siga hoy en día utilizando alguna de las Metodologías, Técnicas y Herramientas entonces seleccionadas, tras el arduo trabajo de prospección del mercado, prueba, y, desde luego, religioso pago a los proveedores que entonces se hizo. Siempre hubo una enorme reticencia (resistencia, diría yo) al cambio por nuestra parte. Nosotros, los adalides del cambio, siempre que sea aplicado a los demás, nos negábamos sistemáticamente (y nos seguimos negando, por lo que sé) a cambiar nuestros usos y costumbres, a aceptar que alguien, aunque sea uno de nosotros mismos, venga a decirnos cómo hacer mejor nuestro trabajo. Porque... somos especiales, raritos, únicos, nuestros Sistemas son sofisticadísimos, igual que nosotros mismos lo somos, nuestras corbatas son las más bonitas, y bla, bla, bla... ¡Venga ya! La verdad pura y dura es que hemos fracasado rotundamente en mecanizar al, supuestamente, Departamento más fácil de mecanizar: Nosotros mismos. La versión 1.0 nos ha salido rana. Nunca funcionó de verdad, ni siquiera cuando estaba recién implantada. Ayudó poco y entorpeció mucho, o la entorpecimos nosotros, que también. Y feneció de inanición. Ahora estamos en plena versión 2.0: cambio de paradigma (Orientación a Objetos), cambio de Herramientas, de Tecnología... ¿Seremos esta vez capaces de dotarnos a nosotros mismos de un Sistema Informático que cumpla con las expectativas de cualquier Sistema Informático de los que amorosamente entregamos a nuestros Usuarios? La respuesta está en vuestras manos, amigos. Y conmigo no contéis esta vez, que yo ya fracasé cuando fue mi turno.

# RECONSIDERACIONES

Lamento tener que detener aquí el texto. No es poca cosa lo que queda afuera:

* El cómo los informáticos solucionaron la brecha entre las cantidades ígneas de recursos que implicaba el Sort de millones de registros en los ordenadores (la búsqueda del algoritmo más eficiente, el ordenamiento externo, la herencia de la mecánica secuencial, etc.).
* Las razones del nacimiento del concepto del Data Warehouse, las estrategias que se usaron para vender conceptualmente algo antes de que exista, el recupero de tecnologías y propuestas en busca de su implementación, los problemas de definición, el enfoque de las bases de datos relacionales en contradicción con los datamarts, la necesidad establecer una metodología de diseño.
* Cómo y por qué los Acuerdos de Capital de Basilea II tuvieron un impacto tan decisivo para la creación de una tecnología tipo Business Intelligence, lo que luego fue el espaldarazo para el Data Mining.
* La carrera de los sistemas operativos y su relación con la adopción en las grandes ligas del PC.
* La crisis de los '90 y su impacto en la búsqueda de interfaces que permitan comunicar las nuevas aplicaciones unas con otras y a su vez, con los servidores y el recupero de datos desde las monstruosas mainframes.
* La ventana batch, desarrollada en los 70, como mecanismo aún firme de consolidación de ingentes montos de datos.
* Las tecnologías y las razones comerciales, sociales, financieras, que catapultaron el encumbramiento de la Internet a partir de la burbuja punto com luego del cambio de milenio.
* Uf... mucho más...

¿Y por qué he dejado todo esto afuera? Como escribí al principio: temo estar entregando a la cátedra una extensión que ni quiere ni ha pedido. Además, hacia los 90's el autor del libro empezó a desempeñarse más bien como consultor que como programador, y por eso su análisis se enfoca en razones que empujan a la adopción de ciertas tecnologías por sobre otras (quizá las verdaderas razones). Hoy, por ejemplo, se sobre-simplifica con aseveraciones como "la aparición de este procesador posibilitó que se desarrollaran aplicaciones..., etc.". A veces, la necesidad de la aplicación vino antes que la del procesador, y esa necesidad fue planteada antes por otras necesidades (por ejemplo, la caída de los mercados debido a una crisis del mercado común que obligó a los bancos a salir a ofertar sus productos online; creo haber dejado un ejemplo similar con respecto a las terminales financieras con las que se proveyó a las oficinas bancarias).

Solo quiero manifestar que espero haber dejado en claro que:

* Esa generación del '70 fue maravillosa y genial. Sigo sosteniendo que sentaron las bases de la informática moderna.
* Que mucho de sus logros hoy no son debidamente apreciados. Se está produciendo un alejamiento entre las respuestas que ellos dejaron y el conocimiento de los problemas que con esas respuestas resolvieron.
* Que no parece que se enseñara hoy Estructurada en los términos de qué resuelve, de qué resolvió en el pasado. Tampoco parece que haya un claro camino hacia una metodología de desarrollo que contemple los distintos escenarios. Y es un trabajo que se hizo, ciertamente...

¿Cuál es el análisis prospectivo? Si bien está muy bien expresado en el texto, queda claro, por ejemplo, que no es seguro hablar de una evolución "natural" de la tecnología. A veces (en el libro hay algunos ejemplos bien concretos, no recuerdo si he dejado alguno) la mejor tecnología simplemente no "gana la carrera" de la adopción popular. ¿Dije popular? Sí, popular. Les aseguro que algo que no se vuelve popular está condenado a desaparecer, a ser discontinuado (que es prácticamente lo mismo). ¿Hablo del hardware? No tengo en mente primariamente eso. Tengo en mente las metodologías, los procedimientos, el conocimiento, los frameworks, ¡incluso los lenguajes!

Este trabajo se ha enfocado en el relato de lo que tiene que ver, principalmente, con el desarrollo de las aplicaciones. Y que la PC pasara a ser la norma, ¿cómo no va a influir? Porque digo, es bastante diferente hacer un programa para un sistema Unix de una Workstation de IBM que una SPA (single page application) para una inmobiliaria que quiere solucionar un desmadre que están teniendo con las cobranzas.

Eso, ¿qué nos enseña? A ver el futuro de otra manera. Mejor dicho, a entender el cómo otros nos ven a nosotros, los informáticos. Que somos parte de un proceso de soluciones y propuestas, que somos los técnicos que usan de herramienta el cómputo (computadora) o el ordenamiento (ordenador). Y ¿qué es esta herramienta que usamos? Una máquina, propiamente dicha, no es. No tiene partes mecánicas. Es una herramienta que entiende de ceros y unos. Nada más.

Hago extensivo este razonamiento a los celulares. ¿Quién se habría imaginado hace 30 años que un teléfono iba a terminar siendo un centro de entretenimiento portátil? ¡Un teléfono! ¿O que una computadora iba a ser la forma de concertar citas amorosas con desconocidos? ¡Un ordenador, desde el que el oficial de crédito del banco consultaba nuestros saldos! ¿Cómo se llegó a esto? Pues bien, en base a las necesidades sociales (venga que tal cosa ni a Zuckerberg se le ocurrió...), a las económicas, a las inquietudes psicológicas de la masa, a los desvaríos de las inversiones, etc.

Los informáticos no son los únicos seres en esta tierra. Otros propusieron soluciones. El que tenía un martillo en la mano, las propuso. El que tenía masa laboral, las propuso también. El que tenía muchísimo dinero, las propuso a su manera. Todos con sus herramientas propusieron sus mejoras, sus soluciones. Y el que tenía un martillo en la mano vio que los problemas tenían forma de clavo. Pero la que entre todos elegimos son las opciones que los informáticos pusimos sobre la mesa. ¿Por qué? Por la herencia que recibimos del Siglo XX, siglo en el que el mayor valor pagado fue la información. Y el camino más eficiente y apropiado para conseguir información del dato es, sin duda, la informática, ¿no? Las bases de datos, la minería de datos, las proyecciones estadísticas, la inteligencia artificial, la investigación montada sobre súper ordenadores...

Hoy solo podemos ver que un adolescente considera un gran regalo un teléfono (un celular, mejor dicho, cosa que un adolescente, 30 años atrás, hubiera considerado extraño: un teléfono como presente de cumpleaños...) o que muchos consideran una gran cosa que un auto tenga una computadora de a bordo (cosa que hace 30 años muchos hubieran considerado como una cosa estrafalaria y sin sentido).

Entonces, ¿cuál es el futuro? El que nosotros nos fabriquemos en base a las soluciones o propuestas que ideemos ante las situaciones que sobrevengan. Hoy es el momento de las apps debido a una pandemia imposible de anticipar. Y si de repente una vieja tecnología nos viene bien en la propuesta, pues resurgirá. Me hubiera encantado incluir el capítulo del libro que habla justamente de esto, del surgimiento en su momento de una cosa abandonada como fueron los procesadores vectoriales de los '60, que resucitaron en auge en los '90, y que hace unos años IBM, Toshiba y Sony resucitaron con su microprocesador Cell para consolas de video juegos (porque, venga, que hoy cualquier Millenial no puede desconocer lo que es una Play Station...).

¿Y en cuanto al desarrollo de software? Viendo el ascenso de los lenguajes en boga y el ostracismo al que se enviaron tantos otros, la tendencia claramente favorece a lo declarativo sobre lo procedural, a la aplicación sobre el sistema, al framework sobre la librería. Las tendencias marcan una balanza que, de parte de las masas, se inclina hacia lo online más que hacia lo desktop. Y en esto, en lo cloud, ¡claro que tiene que ver el espaldarazo que las empresas le dan! Empresas que sienten sus datos más protegidos y sus necesidades de hardware mejor atendidas: PaaS, IaaS, MBaaS, NaaS... y si la respuesta desde la programación es la aplicación serverless, ¡pues mucho mejor! Por algo las PWA's vienen pisando fuerte... ¡uf! ya se me acabó la tinta, por mucho que me gustaría seguir comentando esto.

Pero bueno, no se preocupen, no se pierden de mucho, son meras apreciaciones mías, nada más.

Que tengan buenas noches.

1. Conjunto de estudios que se llevan a cabo sobre un tema o situación particular, a modo de determinar lo que ocurrirá de forma anticipada. [↑](#footnote-ref-1)
2. Vaya... [↑](#footnote-ref-2)
3. También conocido como diagrama de flujo. [↑](#footnote-ref-3)
4. Esos saltos, que se dibujaban con líneas que iban de una estructura a otra del diagrama de flujo, eran instrucciones de bifurcación, es decir, los GOTO's. [↑](#footnote-ref-4)
5. Las hojas de codificación eran unas hojas gruesas, como las que se usan para artes plásticas en la primaria, de tamaño similar a esas. Se las leía en forma apaisada. Arriba tenían espacio para poner el nombre o número de hoja o lo que sea. Luego, a la izquierda, iban numeradas en columna (un número sobre otro) y, a cada número, una línea punteada para escribir el código. Es decir, ese block de hojas era, ni más ni menos, nuestro editor de código, nuestro VSCode. [↑](#footnote-ref-5)
6. Se refiere a que era una sola enorme rutina. [↑](#footnote-ref-6)
7. Es decir, esas tarjetas perforadas eran tu código fuente. [↑](#footnote-ref-7)
8. Un ejemplo moderno, corriente y a la mano se ve en los algoritmos recursivos, que puede que no den errores y sean formalmente correctos, pero no hagan lo que se quería que hagan. [↑](#footnote-ref-8)
9. El libro se escribió en el 2009. A la fecha (2020) IBM ya sacó unos modelos más pero que siguen perteneciendo a la misma serie. [↑](#footnote-ref-9)
10. *Palabras del autor*: Como dije, no se conocen para z/OS. No digo que no los haya, sino que de momento no se conocen, así que la seguridad ante intrusiones es máxima. Esto es debido a que, por su arquitectura interna, es bastante difícil que un hacker pueda encontrar una vulnerabilidad explotable en el Sistema. ¡Ya que ya nadie enseña Ensamblador del z/Series!, exceptuando a la propia IBM, que cobra una fortuna por ello. Y recordad, Aplicaciones escritas en los años 80 (cuando no había prácticamente ninguna alternativa importante a los mainframes de IBM) siguen funcionando hoy en día sin problemas, incluso sin siquiera compilar de nuevo los programas, lo que les otorga un Retorno de la Inversión realmente elevado. [↑](#footnote-ref-10)
11. El "batch" es un concepto importantísimo dentro de la informática "en serio". En el libro hay todo un capítulo que trata sobre la ventana batch, pero en honor a la brevedad, lamentablemente debo dejar afuera el tema. Solo diré que el batch es un proceso de total relevancia actual y que tiene que ver con la actualización de carga de datos o de aplicaciones que gobiernan los procesos. [↑](#footnote-ref-11)
12. Un amigo mío (también de la época de los dinosaurios, como yo) comenta siempre que “XML es un standard muy completo, que sólo tiene un fallo: ¡No tener PICTURE!!” [↑](#footnote-ref-12)
13. Vuelvo a repetir lo señalado en alguna nota anterior. El proceso batch es importantísimo y crítico, y al cual el autor del libro le dedica todo un capítulo, que no será expuesto aquí, en este trabajo. [↑](#footnote-ref-13)
14. Téngase en mente que estamos hablando de una época previa al paradigma estructurado. Con esto quiero decir que la programación actual tiene en su genética las técnicas que ya se usaban en la era del espagueti. [↑](#footnote-ref-14)
15. Es decir, en criollo, en gaucho. [↑](#footnote-ref-15)
16. La mención del sort (ordenamiento) parece trivial. Ya decidiré si incluir el capítulo en donde el autor explica lo crucial, en términos informáticos, del sort. Solo adelantaré que, en todo montaje de sistemas, es condición sine qua non el cálculo de carga de procesamiento que el sort implica al benchmark del sistema. Ésta es una de las razones por la que elegí este libro, porque lidia con entornos de trabajo reales. Solo adelantaré que el ordenamiento de los ordenadores (de allí el término), cuando tienen que operar con millones de registros (de la era relacional) implica una de las demandas de procesamiento más grandes a las que son sometidos los sistemas informáticos como un todo. [↑](#footnote-ref-16)
17. Sigue siendo reciente la respuesta de JavaScript de proponer soluciones para el fetch asincrónico. [↑](#footnote-ref-17)
18. En el libro hay un capítulo dedicado a esto pero que no se verá aquí. Tiene que ver con las metodologías de diseño de aplicaciones, y este es un aspecto que sinceramente no sé si a la fecha ya ha sido resuelto, creería que no. [↑](#footnote-ref-18)
19. De ahí viene que, en los manuales de estilo, sigan pidiendo que la codificación trate de no exceder las 80 columnas (cosa que además de buena práctica, no tiene a veces mucha excusa para no hacerse porque, a fuerza mayor, se hacía con el *verbose* Cobol. [↑](#footnote-ref-19)
20. Lo que constituyó una habilidad asombrosa. No encuentro mejor ejemplo de análisis prospectivo. Y ejemplos así, hay muchos en el libro. De adelantados. De gente que fue capaz de ver lo que se venía. Es una pena realmente no tener el espacio suficiente para poder trabajar el libro de pe a pa... [↑](#footnote-ref-20)
21. Los párrafos que vienen relatan las perplejidades y los obstáculos que supuso la adopción del paradigma estructurado. [↑](#footnote-ref-21)
22. Justamente la razón por la que elegí este libro: no he visto otro en el que se relate cómo se resolvieron problemas de cara a hechos reales. Yo debo tener alguna anécdota personal al respecto, pero no para estructurada, sino para funcional. [↑](#footnote-ref-22)
23. Recuérdese que en Cobol, EL LENGUAJE por antonomasia de la época, el flujo era secuencial, orden a orden, el cual terminaba cada línea con un punto y al siguiente. [↑](#footnote-ref-23)
24. No se refiere a la codificación, sino al diseño del algoritmo. La propuesta no usaba un diagrama de flujo, sino una especie de pseudo código que definía las estructuras con llaves, tipo árbol, y a veces, para cumplir formalmente con el método, esas llaves quedaban vacías... [↑](#footnote-ref-24)
25. Este "olvido" de patentar su aplicación (bueno, no es tan simple, pero esa es otra historia) fue determinante para que hoy exista la PC sobre la que usted ahora mismo lee estas líneas. [↑](#footnote-ref-25)
26. Este capítulo lo saqué del práctico por la simple razón de su extensión. Pero lamento hacerlo porque versa sobre el advenimiento de las bases de datos relacionales y cuenta (haciendo un paralelismo entre lo que significó el cambio desde el paradigma "espagueti" al estructurado) el cambio desde el paradigma secuencial al relacional. Obviamente, señala claramente que cómo interactuó la tecnología de los dispositivos (hardware) para implantar esas capacidades. La ampliación de esas consideraciones que llevaron a las decisiones que encaminaron y decantaron a favor de unas tecnologías sobre otras está en otro capítulo que tampoco incluyo, pero en el que el autor explica claramente lo que demanda el trabajo de un ordenador (computadora), las demandas a nivel hardware y el por qué de la búsqueda de algoritmos eficientes en entornos "serios" (de informática de grandes volúmenes). [↑](#footnote-ref-26)