Capítulo 5

La Tecnología

Un mundo como el moderno, en donde casi todo se concibe a la luz de la tecnología, en donde el hombre pasa a ser una presa fácil de las grandes fábricas desde donde lo parasitan argumentándole comodidad pero estableciéndole reglas en su vida social, reglas basadas precisamente en la misma tecnología, es un mundo en el cual se hace imposible no hablar de ella para cualquier libro y mucho mas aún si se trata de un libro técnico.

Sería pues imposible ignorar la tecnología cuando lo que se busca al hablar de algoritmos computacionales es aprovecharla para poder lograr, de la manera mas eficiente, los objetivos que se hayan propuesto. El computador como herramienta tecnológica nos brinda su velocidad para que en unión con nuestros algoritmos se puedan obtener velozmente resultados que de otra forma tomarían muchisísimo tiempo, sí, así como está escrito muchisísimo tiempo. Encontrar un dispositivo o un aparato que puede trabajar en términos de millonésimas de segundo es muy útil y mucho mas si podemos aprovechar dicha velocidad para nuestra conveniencia.

Ya en su momento explicábamos que luego de que se ha concebido apropiadamente un algoritmo es decir luego de que hemos comprobado a través de una “prueba de escritorio” que el algoritmo realmente sí nos permite alcanzar el objetivo propuesto entonces pasamos a una segunda etapa que corresponde a la intervención de la máquina en la solución del problema inicial. Esta etapa inicia con la transcripción que no es otra cosa que reescribir nuestro algoritmo pero en términos de un determinado Lenguaje de Programación. Pues bien, el Lenguaje de Programación lo podemos definir como ese puente perfecto que permite que el computador ejecute lo que nosotros habíamos concebido como un algoritmo (y además que lo haga a altas velocidades).

Un Lenguaje de Programación es, técnicamente hablando, un conjunto de instrucciones que son entendibles y ejecutables por un computador. No podemos esperar, por lo menos no por ahora, que el computador ejecute lo que nosotros concebimos como algoritmo (aunque sería lo óptimo) y

por eso debemos incorporar al desarrollo técnico un paso o un conjunto de pasos mas y son lo que involucran los Lenguajes de Programación.

Esto significa que si nosotros hemos desarrollado un algoritmo computacional que es muy útil pues de nada nos va a servir si no conocemos las reglas sintácticas de un Lenguaje de Programación, si no escribimos el algoritmo con esas reglas, si no contamos con el compilador del Lenguaje en el cual vamos a trabajar y si no sabemos interpretar los errores. Por lo tanto el contacto técnico que vamos a tener con el computador va mucho mas allá de desarrollar solo el algoritmo ya que es nuestra obligación hacer realidad el algoritmo a través del Lenguaje de Programación.

Para dar un espacio de trabajo claro con los Lenguajes de Programación, éstos se encuentran categorizados en dos clases:

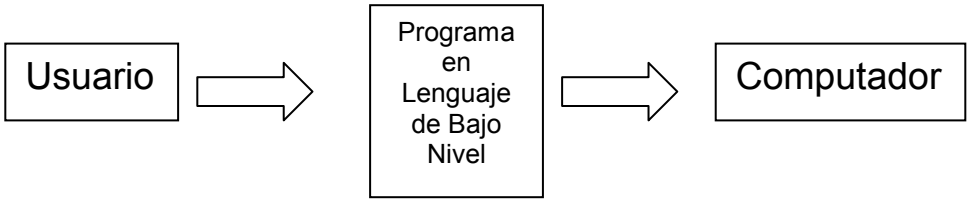
Lenguajes de Bajo Nivel

Son aquellos lenguajes en los cuales sus instrucciones son para nosotros complejas de entender pero que son extremadamente sencillas para ser entendidas por el computador. Tal es el caso del Lenguaje Assembler según el cual las instrucciones está basadas en una serie de mnemónicos que no siempre facilitan la transcripción. Este Lenguaje es el lenguaje verdadero de los computadores. Esto quiere decir que internamente lo único que entienden los computadores realmente son instrucciones del Lenguaje Assembler. La programación a bajo nivel fue la que se utilizó en las primeros años de la Historia de la Programación. Cada vez los programadores tenían que ser mucho mas especializados dada la complejidad de entendimiento de las instrucciones utilizadas en este lenguaje. No se podía desconocer el inmenso poderío de los lenguajes de bajo nivel pero tampoco se podía desconocer la complejidad de su estructura.

Si usted encuentra una instrucción como la siguiente

mov bp, sp

No es muy fácil deducir qué hace o para qué le serviría en el desarrollo de un programa. Sin embargo puede tener la seguridad que esta instrucción es perfectamente clara para el computador ya que su interpretación es inmediata. De esta manera la programación con lenguajes de bajo nivel puede representarse con el siguiente diagrama de bloques



Lenguajes de Alto Nivel

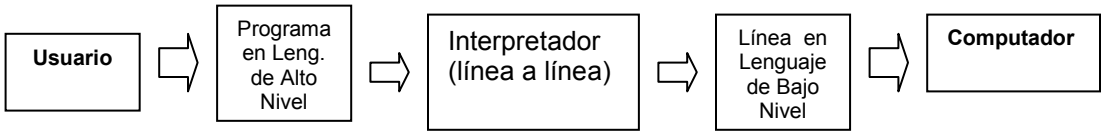
Precisamente pensando en estas desventajas de la programación a Bajo Nivel y sabiendo que quien realmente desarrollaba los programas era el ser humano y no la máquina y que la máquina gracias a su velocidad era solo la encargada de ejecutar la orden que se le diera, se pensó en crear unos Lenguajes de programación que fueran mas entendibles al ser humano o sea unos Lenguajes de Programación en donde las órdenes fueran tan sencillas de comprender que programar se convirtiera en algo realmente sencillo.

Estos Lenguajes fueron llamados Lenguajes de Alto Nivel sabiendo que de todas maneras el computador necesitaría otro programa que tomara lo que se escribiera y lo convirtiera a lenguaje de bajo nivel (ya que no podemos olvidar que éste es el verdadero lenguaje del computador). De allí surgió la idea de los *Interpretadores* y los *Compiladores*.

Lenguajes Interpretados

Son aquellos lenguajes de programación en donde existe un programa interpretador que no es mas que un programa que "coge" nuestro programa y lo convierte línea a línea a Lenguaje de Bajo Nivel y así mismo lo va ejecutando (o sea línea a línea). Estos lenguajes tenían de inconveniente que si el programa tenía un error en una de las últimas líneas solo hasta cuando el interpretador llegaba hasta allá era que se detectaba, luego de que ya se había ejecutado todo un gran bloque de instrucciones.

Sin embargo, y para los inicios de la programación a través de Lenguajes de alto nivel, esta era la solución precisa, en esos tiempos, para poder programar con lenguajes mas entendibles al ser humano que al computador. De esta manera la utilización de lenguajes interpretados tomó mucha popularidad por las grandes facilidades que le brindaban al programador que ya no tenía que invertir la mayor parte de su tiempo en especializarse en entender instrucciones de alguna manera complejas. Podríamos pues esquematizar la utilización de lenguajes interpretados con el siguiente diagrama de bloques



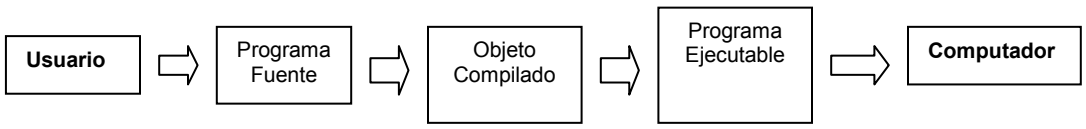
Se incorpora pues un elemento adicional como es el interpretador (línea a línea) que le facilita el trabajo al programador pues éste ya puede utilizar instrucciones mas entendibles como ***print*** que le representará Imprimir y que con solo leerla da una idea de que se trata de escribir. Esa es precisamente la diferencia exacta entre las instrucciones de un Lenguaje de Bajo Nivel y un Lenguaje de Alto Nivel y radica, dicho de una manera sencilla, que por el hecho de que las instrucciones son mas entendibles los programas y el proceso de transcripción es mas sencillo y menos exigente técnicamente.

Sin embargo la dificultad presentada por los Lenguajes Interpretados en cuanto al hecho de que convertían línea a línea el programa y así lo ejecutaban estableció el riesgo de que en programas comerciales, en donde es fácil encontrar 15000 o 20000 líneas en un programa, era demasiado costoso a nivel empresarial y demasiado riesgoso a nivel técnico saber que en cualquier momento por un error determinado el programa podía ser interrumpido (a abortado como técnicamente se le dice) debido a que el interpretador iba revisando línea a línea. Ello llevó a pensar en un esquema mucho mas práctico y menos arriesgado tanto para programadores como para empresarios.

Lenguajes Compilados

Son aquellos lenguajes en donde un programa llamado compilador toma TODO el programa que hemos escrito (que normalmente se denomina Programa Fuente), lo revisa y solo hasta cuando esté completamente bien, solo hasta allí lo convierte a su equivalente en Lenguaje de Bajo Nivel para ser ejecutado por el computador. De esta manera se reducía el riesgo de evaluación que representaban los Lenguajes Compilados y se podía saber si TODAS las instrucciones del programa eran correctas. De hecho era de suponer que, bajo condiciones normales, el programa entonces no se abortaría cuando estuviera en su momento de ejecución (cosa que no es 100% cierta). Lo que sí se puede decir es que la cantidad de interrupciones de los programas elaborados con Lenguajes Compilados se redujo notoriamente frente a las interrupciones que sufrían los programas elaborados con Lenguajes Interpretados.

De allí que cuando se compila el programa fuente se genera un segundo programa (que solo es entendible por el computador) conocido como Objeto Compilado y cuando se va a ejecutar entonces este programa es “organizado” de manera que cada instrucción sea perfectamente entendible y ejecutable por el computador. A este nuevo programa se le conoce como Programa Ejecutable. De esta manera podemos resumir el proceso de utilización de los Lenguajes compilados a través del siguiente diagrama de bloques:



De aquí surgen entonces los pasos que inicialmente se habían explicado en cuanto a la metodología para la solución de un problema cuando éste se trata de un algoritmo computacional. En la actualidad podríamos decir (casi con toda seguridad) que los Lenguajes de Programación que existen en el mercado son Lenguajes Compilados por su facilidad de uso y por sus características de funcionamiento.

Ya podemos también concluir que un Lenguaje de Programación está diseñado para facilitarnos el trabajo de hacer que el computador haga algunas cosas por nosotros. Entonces se pregunta uno porqué existen tantos Lenguajes de Programación..? La respuesta es muy sencilla y se puede establecer bajo dos criterios:

La primera razón es estrictamente comercial ya que detrás de cada Lenguaje de Programación existe una gran empresa vendiéndole al mercado informático su producto (o sea el compilador y sus utilidades). No en vano vemos como empresas como Microsoft se han dedicado al desarrollo de Lenguajes de Programación (entre otros productos) y son en la actualidad una de las empresas mas grandes y mas rentables del mundo. El ánimo competitivo a nivel empresarial en el mundo informático y mas exactamente en lo que se refiere al desarrollo de Lenguajes de Programación es lo que ha hecho que exista una gran cantidad de Lenguajes de Programación.

La segunda razón es mas técnica y menos comercial ya que no se puede desconocer que los Lenguajes de Programación también se han desarrollado para lograr de una manera mas fácil y eficiente unos objetivos específicos. De esta manera algunas empresas se han especializado en colocar a disposición de los programadores un conjunto de instrucciones que le permitan lograr ciertos objetivos concretos de una forma muy sencilla y sin tanto rigor técnico. De allí que encontramos que el Lenguaje **Fortran** buscaba facilitar la traducción de fórmulas matemáticas, el Lenguaje **Basic** buscaba entregarle al alumno de programación las herramientas para que hiciera sus primero pinitos en programación, el lenguaje **Logo** buscaba iniciar al niño en el mundo de la programación de computadores, el Lenguaje **Pascal** buscaba brindar los mismos elementos del Basic pero con una filosofía y un pensamiento estructurado, el Lenguaje **Cobol** buscaba facilitar el manejo de archivos y en general de memoria secundaria, etc.

No con lo anterior quiero decir que esos lenguajes solo sirvan para lo que está allí consignado, lo que sí quiero decir es que ESA es la tarea que mas facilitan, cosa que para un programador es supremamente importante porque de allí depende, en gran parte, que su tarea sea realmente sencilla o no.

Errores en un Programa

Cuando se dice que un compilador revisa un programa se está dando una definición muy amplia de lo que realmente es esa revisión. Primero que nada vamos a revisar los dos tipos de errores que

puede tener un programa. Ellos son los errores Humanos y los Errores detectables por un Compilador.

Errores Humanos

Son todos aquellos errores que dependen exclusivamente de la participación del ser humano en el proceso de escritura de un programa (a partir de un algoritmo) obviamente. Es evidente que son errores que no son detectados, en estos tiempos, por un compilador ya que dependen exclusivamente de la visión, práctica, experiencia y conocimientos que tenga un programador. Los errores humanos son de tres tipos:

Errores de Concepción

Este es el tipo de error que se presenta cuando el programador cree que tiene el objetivo claramente identificado y entendido y resulta que no es así (pero él no lo sabe) con lo cual es evidente que finalizado el programa seguramente habrá logrado algo totalmente diferente a lo que inicialmente necesitaba lograr. Por ejemplo un programador quiere implementar en un computador un programa que realice cierto tipo de liquidación de un estado de cuentas de un cliente. Averigua y se entera de que en la empresa para la cual trabaja la liquidación de los clientes se hace de una manera diferente a la que hacen las demás empresas ya que en ésta se tiene en cuenta el tiempo de vinculación con la empresa para reducir su deuda final. El programador investiga y creyó entender cómo realmente lo hacían cuando en realidad no era así pues no captó la utilización de un factor que de manera autocrática es establecido por el criterio del gerente. El resultado final es una liquidación que a pesar de ser muy detallada y muy completa no coincide con lo que realmente se necesitaba en la empresa.

Pareciera ser muy poco frecuente este error y resulta ser lo contrario pues yo le pregunto lo siguiente *Quién le garantiza a usted que lo que usted cree que entendió es porque realmente lo entendió..?* Pues nadie porque no es fácil saber si alguien captó realmente lo que le quisimos decir. En este sentido es importante que se tenga en cuenta que debemos buscar muchos métodos para demostrar, sobre todo a nosotros mismos, que sí entendimos lo que queríamos entender. Para el caso de ese programador del ejemplo, qué debió hacer él antes de empezar a programar..? Pues debió haber solicitado los datos completos de varios clientes y haber realizado MANUALMENTE la liquidación. Solo de esta manera hubiera sabido que no le era claro aquel factor que colocaba el gerente o por lo menos hubiera encontrado un camino para aproximarse a dicho factor.

De manera que este error podemos decir que es el más difícil de detectar si no establecemos los mecanismos para ello sobre todo porque el computador en ninguno de sus procedimientos va a detectarlos.

Errores Lógicos

Son los errores que se presentan cuando no se ha comprobado apropiadamente a través de la “Prueba de escritorio” la efectividad de un algoritmo. Estos errores solo son detectables a través de dicha Prueba, el computador nunca nos va a hacer recomendaciones dentro del campo de la

lógica. Por eso es tan importante que sepamos que luego de reconocido el objetivo y de desarrollado un algoritmo es imprescindible realizarle una prueba de escritorio. Cuando ésta no se realiza es muy posible (extremadamente posible) que nuestro algoritmo no satisfaga el objetivo propuesto.

Por ejemplo en una empresa en donde primero se realiza un determinado descuento sobre el valor bruto de una compra y luego se liquida la factura para el cliente será importante que solo en ese orden el programador de turno lo realice porque si primero liquida la factura del cliente y luego le aplica el descuento los resultados podrán ser (y lo serán) completamente diferentes a los esperados y en ese momento el computador no le va a decir que fue lo que hizo malo o lo que dejó de hacer.

Errores de Procedimiento

Son los errores que se presentan cuando se tiene claro el objetivo y se ha desarrollado un algoritmo aproximadamente bien pero no se realiza bien la "Prueba de Escritorio", es decir, se hace la Prueba de Escritorio pero se hace mal y no nos damos cuenta y además de ello quedamos convencidos que el algoritmo quedó bien. Por eso no solo es importante que se realice la prueba de escritorio para evitar los errores lógicos sino que además se realice bien, apegada a las instrucciones. Recuerde que lo único que necesita para realizar una buena Prueba de Escritorio es que usted mientras la esté realizando no "razone" o sea que actúe como si usted fuera un computador (que no piensa) y de manera autómata realice y obedezca una a una las órdenes establecidas en el algoritmo.

Esa será la única forma de usted estar seguro de que realizó bien la prueba de escritorio porque éste es otro de los factores que no es fácil de detectar ya que muchas veces podemos hacerlas y no tener la certeza de si lo hicimos bien o no. Por eso la única forma que usted va a tener a la mano para estar completamente seguro de que si la hizo bien es no actuar como un ser humano sino como un computador. Si usted encuentra en alguna parte del algoritmo la orden Escriba que 5+3 es igual a 9 no lo razone sencillamente escríbalo y notará al final si eso le ocasionó un error o no. La realización efectiva de Pruebas de Escritorio en algoritmos computacionales será vista en detalle en el capítulo siguiente en donde veremos cuál es la metodología para tener toda la certeza de que nuestra Prueba quedó bien.

Sin temor a equivocarme puedo decirle que son los errores del Ser Humano los mas difíciles de detectar en el desarrollo de un algoritmo dado que dependen solo de la metodología y el orden que para solucionar el problema tenga el mismo ser humano. También podríamos decir que dichos errores son los mas complejos porque su solución se remite a cambiar una concepción acerca de un problema o incluso realizar algunos ajustes a la lógica de programación para acercarnos mas a la lógica computacional y abandonar un poco la lógica humana. También podemos decir que los errores humanos son los mas difíciles porque ningún aparato nos dice en donde están. Tenga pues en cuenta que solo de usted va a depender que no tenga que cometer estos errores ya que al final resultan ser los mas costosos si se analizan desde una óptica estrictamente empresarial.

Errores detectados por un Compilador

Son los errores mas sencillos ya que los compiladores modernos no solo nos dicen cuál es el error sino que además nos orientan en donde puede estar dicho error. Es muy importante que sepa que en cuanto a los errores detectados por un compilador usted deberá saber dos cosas:

Qué significan.- Debido a que normalmente los compiladores son desarrollados fuera del país y los errores salen como un aviso en inglés. Saber qué significan simplemente es conocer su traducción literal y no mas.

Qué representan.- Esto es lo mas importante en un aviso de error generado por un computador ya que la representación de un error es lo que realmente nos va a permitir corregirlo apropiadamente. Tomemos un breve ejemplo, usted ha realizado un programa en el que ha utilizado variables de las cuales se le olvidó declarar una variable que dentro del programa usted ha llamado conta1. En algún momento del programa usted asigna el valor 35 a dicha variable. Cuando lo compila le sale el siguiente error

Impossible assign value in variable conta1

Qué significa..? Imposible asignar valor en la variable conta1 (no olvide que el significado es la traducción literal).
Qué representa..? Que el computador no puede asignar el valor porque la variable en mención (o sea conta1) no está declarada y por lo tanto para él no existe.

La familiarización con el significado y la representación de los errores de un determinado Lenguaje de Programación se da solamente en la medida en que usted realice muchos muchos programas en dicho Lenguaje. Este puede ser un buen termómetro para determinar si una persona realmente tiene experiencia en un determinado lenguaje de programación o no.

Pasemos pues a describir los dos tipos de errores que genera un compilador cuando está “revisando sintácticamente” un programa. Estos errores son

Errores de sintaxis

Son las omisiones que cometemos cuando transcribimos el programa. Es normal que se nos olvide cerrar un paréntesis que hemos abierto o declarar una variable. El compilador nos dice a través de un aviso cuál es el error y nos ubica aproximadamente en donde está. Cabe anotar que estos errores no dejan que el programa sea ejecutado.

Errores de Precaución

Son recomendaciones técnicas que el compilador nos hace para que el computador nos garantice el correcto funcionamiento del programa. Podría decirse que no son errores como tales pues el programa puede ejecutarse aún a pesar de tener errores de precaución con la pequeña diferencia de que es muy posible que los resultados finales no sean los que esperamos.

Como podemos distinguir cuando el compilador nos indica un error de sintaxis o un error de precaución..? Pues sencillamente porque en pantalla cuando nos aparezca el error al lados dice si es Syntax Error ó Warning Error que significan Error de Sintaxis o Error de Precaución respectivamente. Por esta razón podemos asegurar sin temor a equivocarnos que los errores detectados por un compilador son los mas sencillos pues el mismo compilador nos dice cuáles son y, mas o menos, en donde están. Nuestro único aporte es la interpretación de estos errores.

Desarrollo Histórico de la Programación

Como todas las ramas del conocimiento humano, la programación también ha ido avanzando haciendo que ésta sea cada vez mas simplificada para el programador y brindando día a día mas herramientas técnicas que permitan la utilización de los computadores de una manera sencilla y simplificada. También se han experimentado algunos cambios en cuanto a la concepción del mundo de la programación y sus efectos y utilización en el mundo de la informática.

Cuando comienzan los computadores o lo que en esos tiempos era un computador no existían, como era obvio, las facilidades tecnológicas que hoy existen, razón por la cual el concepto de programación se realizaba a través de circuitos eléctricos y electrónicos directamente de tal manera que los “programadores” de aquellos tiempos eran unos tremendos en electrónica ya que las soluciones las construían con partes como tales. Esa programación se conoció como ***Programación Directa ó Real*** dado que el contacto entre el programador y la máquina era directo y requería un altísimo conocimiento técnico no solo de partes electrónicas sino también de lo que en ese entonces era la programación a bajo nivel.

Poco a poco la tecnología fue avanzando permitiendo, en este campo, que el ser humano tuviera cada vez mas y mejores herramientas de trabajo. Fue entonces cuando se pensó en la programación tal y como se concibe en el día hoy, es decir, permitir que a través de órdenes dadas al computador éste pudiera realizar unas tareas a altas velocidades. Este concepto comenzó a ser trabajado y poco a poco empezaron a surgir en el mercado Lenguajes de programación como tales. Estos Lenguajes permitían realizar un gran número de tareas con la simple utilización correcta de unas instrucciones. La metodología de la utilización de estas instrucciones fue en esos tiempos algo muy libre razón por la cual a esta etapa de la programación se le conoció como ***Programación Libre***.

Bajo esta técnica de programación, la persona que estuviera al frente del computador podía realizar todas las tareas que pudiera o mas bien que el lenguaje le permitiera basado solamente en su lógica propia aplicada a la libre utilización de dichas instrucciones. Aún a pesar de que en principio esta forma de utilizar los Lenguajes de Programación fue la solución para muchos problemas y de que el mundo había comenzado a ser mas eficiente en el tratamiento de la información gracias precisamente a la utilización de órdenes para programar los computadores, los problemas no esperaron para dejarse venir.

Cuando un programador se sentaba con su lógica propia a resolver un problema utilizando las instrucciones que un Lenguaje de Programación le permitía muchas veces (y casi siempre) llegaba a soluciones que solamente él entendía y cuando este programador era sacado de la empresa o se retiraba o se moría entonces la empresa se veía en la penosa obligación de conseguir otro programador que en la mayoría de los casos lo que hacía era volver a hacer todo lo que el primero había hecho pero con su propia lógica quedando la empresa en manos de este nuevo programador y teniendo previsto el gran problema que se originaría cuando éste se retirara o se muriera o hubiera que echarlo.

Fue allí en donde comenzó a pensarse en la Lógica Estructurada de Programación o mas bien se comenzó a pensar que los programas por diferentes que fueran obedecían a una serie de normas que eran común en cualquier algoritmo, término que se comienza a acuñar técnicamente en esa época. A través de muchos estudios se llegó a la conclusión que la lógica de programación se basaba solo en tres estructuras como son las secuencias, las decisiones y los ciclos. Se puso a prueba esta teoría y se descubrió que era cierta pues ningún algoritmo se salía de estas tres estructuras.

Pensar en unas estructuras básicas del pensamiento al momento de la programación facilitó enormemente el hecho de que el programa desarrollado por un programador fuera entendido sin mayores complicaciones por otro. También esta forma de programación restringió el desorden de algunos programadores porque le colocó unos límites a la lógica computacional que era la que había que utilizar cuando se necesitara escribir un programa. A esta forma de trabajo se le llamó ***Programación Estructurada*** que no es mas que la técnica a través de la cual se utilizan los Lenguajes de Programación utilizando las estructuras básicas y permitiendo que los programas sean mucho mas entendibles ya que no son concebidos al libre albedrío del programador sino basado en unas normas técnicas.

Esta técnica de programación comenzó a tomar mucha fuerza en el desarrollo de la programación debido precisamente a que ya un programador podía tomas los programas de otro y entenderlos con muchísima facilidad. Se desarrollaron Lenguajes que permitieran precisamente la sana utilización de estas estructuras y a éstos se les llamo Lenguajes Estructurados. Además dichos lenguajes se podría decir que casi obligaban al programador a no salirse del marco conceptual de las estructuras básicas.

El mundo y el ser humano ávido de soluciones para sus necesidades utilizaron esta técnica de programación estructurada por mucho tiempo sin cuestionarla hasta que las mismas necesidades de programación comenzaron a cuestionar lo que hasta ese momento había funcionado tan perfectamente. Se partió de la teoría que la programación no es mas que una interpretación del mundo real y su simulación a través de un computador por tal motivo se pensó en aproximar

mucho mas los conceptos de programación al mundo real y fue allí en donde se encontró que todo lo que nos rodea tiene unas características y sirve para algo. Por ejemplo un lápiz tiene peso, color, olor, sabor (si se quiere), longitud, espesor, torque, textura y muchas otras características. Al mismo tiempo un lápiz sirve para escribir, para separar una hoja de un libro, para rascarse la espalda, para defenderse de un atraco, para señalar un punto, para dibujar, para manchar a alguien y para miles de cosas mas.

Esta concepción llevó a una gran revolución en la historia de la programación pues se crearon dos vertientes dentro de la lógica de programación: La programación estructurada que ya definimos y la ***Programación Orientada A Objetos*** por medio de la cual se podía modelar el mundo en el computador tal y como es. Su aporte principal era el concepto de objeto. Qué es pues un objeto..? En términos generales un objeto no es mas que un ente informático que tiene características (técnicamente llamadas atributos) y que sirve para algo (técnicamente se dice que tiene unos métodos).

Así se creó pues este concepto y se comenzaría a utilizar los objetos (en programación) que como ya dijimos no son mas que tipos de datos con atributos y métodos propios. Toda una teoría se comenzó a derivar de esta nueva concepción del mundo y se fue aplicando poco a poco en la programación ya que se empezaron a descubrir unas relaciones entre objetos, unas operaciones entre objetos y en general un montón de conceptos nuevos en cuanto a lo que inicialmente no habían sido mas que los objetos. Mientras se desarrollaba esta teoría y se ponía en práctica en muchos de los Lenguajes de Programación comerciales también se seguía utilizando la técnica de programación estructurada pues estas dos técnicas no eran excluyentes. Dependía pues del programador que tuviera una verdadera concepción acerca del problema que quería solucionar la decisión de saber por cuál técnica de programación (programación estructurada o programación orientada a objetos) era mas apropiado resolverlo.

Ello exigía simultáneamente que el programador no solo conociera muy bien los conceptos de programación sino que también conociera muy bien el problema que iba a solucionar y las características de cada una de las técnicas de programación. Tenía que se un profesional integral de la programación pues ahora no solo se necesitaba que supiera de computadores o de electrónica o que se supiera las instrucciones de un simple lenguaje. Ahora tenía que conocer teoría y combinarlas de manera que pudiera llegar a la mejor solución aprovechando la tecnología existente. Debo decir que cuando comenzó a tomar fuerza la teoría de la programación orientada a objetos no todo lo Lenguajes de Programación (o mas bien no todos sus compiladores) estaban acondicionados para que aceptaran la nueva forma de programar.

De esta manera también era necesario que el programador supiera si el problema que iba a solucionar a través de un programa era implementable fácilmente con el Lenguaje de Programación que tuviera a la mano pues debe usted saber que no es fácil inducir la compra de un Lenguaje de Programación (o sea de su compilador) en una empresa cuando todo el sistema de información está basado en otro Lenguaje de Programación. Esta filosofía de programación fue cogiendo mucha fuerza y con ella se fueron fortaleciendo los lenguajes que habían iniciado la aceptación de esas nuevas características. Empresas fabricantes que hasta ese momento habían sido competitivas se convirtieron en verdaderos imperios de la informática. La programación definitivamente había dado un salto impresionante hacia la solución de muchos problemas que eran, en algunos casos, mas complejos de resolver con programación estructurada que con programación orientada a objetos.

Poco a poco algunos fabricantes de Lenguajes de Programación se fueron introduciendo en el mercado y aprovechando las características de la nueva técnica de programación fueron dejando de lado, de alguna manera, la programación estructurada que algunos libros han llamado erróneamente Programación Tradicional. En ese avance tecnológico y con el ánimo de entregar al mercado de la programación mas y mejores herramientas de trabajo se empezó a manejar un concepto muy importante en programación como es el concepto de interfaz. Una interfaz no es mas que la forma como usted puede mostrar la información por medio de algún dispositivo de salida. Ya se sabía que entre mas clara y entendible fuera la información podría decirse que los programas serían mejores ya que lo que finalmente el usuario de un programa necesitaba era que la información que le arrojaba un computador fuera claramente entendible.

Se fue notando pues, por parte de las empresas fabricantes de Lenguajes de computadores, como el tiempo de un programador se iba en su mayor parte en el diseño de las interfaces o sea en el diseño de la presentación de los datos. Por tal motivo se pensó que, en unión con la teoría de programación orientada a objetos y con las herramientas que ella facilitaba, se hacía necesario diseñar lenguajes de programación que facilitaran el diseño de interfaces para que el programador invirtiera su tiempo mejor en el diseño de procesos o de manipulación y tratamiento de datos.

Fue entonces cuando entraron al mercado los Lenguajes Visuales y se incorporó al desarrollo de la programación la ***Programación Visual*** que no es mas que una forma de programar en donde se cuenta con una gran cantidad de herramientas prediseñadas para facilitar, precisamente, el diseño de interfaces. Este tipo de programación ha llevado a que en el mundo de la informática y exactamente en la programación se llegue a unos resultados mucho mas convenientes y mejores a nivel técnico pues en la actualidad se pueden obtener aplicaciones de computador mucho mas entendibles y manejables por el usuario gracias a la filosofía incorporada por la Programación Visual.

A este nivel la programación requería menos conceptos técnicos y mas lógica de programación que era lo que realmente se necesitaba para desarrollar un programa. Es normal ver como una persona con unos modestos conocimientos de computación puede, a través de Lenguajes Visuales, desarrollar aplicaciones verdaderamente útiles y además muy bien presentadas. Lo que poco a poco se fue afianzando fue la necesidad de tener unos conceptos de lógica de programación bien fundamentados para poder aprovechar de una manera eficiente los recursos que la informática le entregaba al computador.

Como se busca modelar con el computador al mundo que nos rodea y en ese avance la tecnología cada vez se ha ido mejorando mas y mas se espera que dentro de muy poco se podrá hablar de una ***Programación Virtual*** en donde el programador pueda ver en tres dimensiones (3D) todo el escenario que necesita para crear sus aplicaciones. Es muy posible que cuando este libro esté en sus manos algunos de estos lenguajes de programación ya estén en el mercado.

Tenga en cuenta que en un país como el nuestro (dependiente de la tecnología que viene del exterior) los avances tecnológicos van un poquito rezagados comparando con los países industrializados dado que es allá en donde se desarrolla la llamada Tecnología de Punta que no son mas que los nuevos avances tecnológicos que le permiten al hombre tener mas y mejores herramientas para aplicarlas en la solución de sus necesidades (y volverse dependiente de ellas).