32

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ENEP - ARAGON

CALIDAD EN LA PLANEACION DE PROYECTOS DE DESARROLLO

TESIS

QUE PARA OBTENER EL TITULO DE
INGENIERO EN COMPUTACION

PRESENTA:
ALFONSO MARTINEZ ORDOÑEZ

EDO. DE MEXICO 2002

PAGINACION DISCONTINUA

Calidad en la planeación de proyectos de desarrollo

Agradecimientos

A Dios:

Por permitirme llegar hasta donde estoy en este momento.

A mi Madre:

Gracias por darme la vida, por apoyarme siempre, por todos tus cuidados, consejos y tu tierno cariño.

A mi Padre:

Gracias por haberme inculcado los principales valores para enfrentar la vida (honradez,sencillez,responsabilidad, etc.), por todo tu apoyo incondicional y por ser el fuerte pilar de la familia.

A la Universidad:

Por brindarme la oportunidad de tener una carrera universitaria y sentirme orgulloso de pertenecer a ella.

A mis maestros:

Por proporcionarme las bases y armas para defenderme ante los retos profesionales, y poder aplicar los conocimientos recibidos en las aulas para el servicio del país.

CONTENIDO

CAPÍTULO 2: CALIDAD DE LOS DESARROLLOS

XII.- CONCEPTOS DE INGENIERÍA DE SOFTWARE
- XII.1.- Confiabilidad
- XII.2.- Utilidad
- XII.3.- Portabilidad
- XII.4.- Eficiencia
- XII.5.- Claridad

XIII.- IMPORTANCIA DEL SOFTWARE
- XIII.1.- Perfil del personal de sistemas
- XIII.2.- El costo del Software
- XIII.3.- Distribución del costo en los proyectos
- XIII.4.- Factores principales en la calidad, productividad y en el costo del Software
  - XIII.4.1.- Capacidad del programador
  - XIII.4.2.- Comunicación en el equipo de desarrollo
  - XIII.4.3.- Complejidad del producto
  - XIII.4.4.- Notaciones apropiadas
  - XIII.4.5.- Control de cambios
  - XIII.4.6.- Nivel tecnológico
  - XIII.4.7.- Nivel de confiabilidad
  - XIII.4.8.- Captación del problema
  - XIII.4.9.- Tiempo y recursos disponibles
  - XIII.4.10.- Habilidades requeridas
  - XIII.4.11.- Facilidades y recursos
  - XIII.4.12.- Formación inadecuada
  - XIII.4.13.- Habilidades gerenciales
  - XIII.4.14.- Otros factores

XIV.- ASEGURAMIENTO DE LA CALIDAD
- XIV.1.- Solicitud de desarrollo
- XIV.2.- Carta de especificaciones
- XIV.3.- Revisiones de Software (Recorridos e Inspecciones)
- XIV.4.- Pruebas de Software
- XIV.5.- Probando unidades de programa
  - XIV.5.1.- Pruebas de Integración
  - XIV.5.2.- Pruebas de Integridad
- XIV.6.- Herramientas automáticas de prueba :
- XIV.7.- Pruebas de aceptación o validación
- XIV.8.- Fase de Liberación
- XIV.9.- Control de calidad en los mantenimientos

- RESUMEN

Capítulo 3: PLANEACIÓN Y ADMINISTRACIÓN DE UN PROYECTO DE DESARROLLO

XV.- FACTORES Y HERRAMIENTAS DE PLANEACIÓN
- XV.1.- Establecimiento de metas y requisitos
- XV.2.- Cascadas de compromiso
- XV.3.- Desarrollo de la estrategia de una solución
- XV.4.- Determinación de Prototipos
- XV.4.- Tipos de Recursos
  - XV.4.1- Recursos de Software
  - XV.4.2.- Recursos Hardware
  - XV.4.3.- Recursos reutilizables

XVI.- PLANEACIÓN INICIAL DE UN PROYECTO DE SOFTWARE
- XVI.1.- Puntos estratégicos para el desarrollo de proyectos (Metas, documentos y revisiones).
- XVI.2.- División del proyecto en estructuras de trabajo.
  - XVI.2.1.- Estructura Funcional
  - XVI.2.2.- Estructura por proyecto
  - XVI.2.3.- Estructura Matricial (Hibrida)
- XVI.3.- Estructura del grupo de programadores

XVII.- ESTIMACIÓN DEL PROYECTO
- XVII.1.- Estimación a través del Juicio Experto
- XVII.2.- Estimación por medio de la técnica DELFI
- XVII.3.- Estimación a través de Estructuras de División de Trabajo (WBS)
- XVII.4.- Estimación del nivel de contratación (Modelo de estimación de Putnam)
- XVII.5.- Estimación de los costos de mantenimiento del Software
- XVII.6.- Herramientas automáticas de Estimación

XVIII.- ANÁLISIS E IDENTIFICACIÓN DEL RIESGO

XIX.- PLANEACIÓN EN EL DISEÑO DE SOFTWARE
- XIX.1.- Conceptos de diseño
  - XIX.1.1.- Abstracción
  - XIX.1.2.- Ocultamiento de Información
  - XIX.1.3.- Estructura
  - XIX.1.4.- Modularidad
- XIX.2.- Arquitectura de software
- XIX.3.- Estructura y diseño de datos
- XIX.4.- Independencia Funcional en el diseño
- XIX.5.- Diseño procedimental o detallado
- XIX.6.- Notaciones de diseño
  - XIX.6.1.- Diagramas de Flujo y formas de representación
  - XIX.6.2.- Diagramas de Cajas
  - XIX.6.3.- Diagramas HIPO (Hierarchy - Input - Proces - Output)
  - XIX.6.4.- Seudocódigo
- XIX.7.- Metodología de diseño

XX.- PLANEACIÓN DE LA CODIFICACIÓN
- XX.1.- Estilo de codificación

XXI.- PLANEACIÓN AL MANTENIMIENTO
- XXI.1.- Tiempo invertido en el mantenimiento.
- XXI.2.- Costos del mantenimiento
- XXI.3.- Planeación de las actividades de mantenimiento
- XXI.4.- Esfuerzo dedicado en el mantenimiento del código fuente

- RESUMEN

CONCLUSIONES

ANEXOS

- Anexo 2.1.- Solicitud de Desarrollo
- Anexo 2.2.- Carta de Especificaciones
- Anexo 2.3.- Pruebas de Integridad
- Anexo 2.4.- Pruebas de Validación
- Anexo 2.5.- Carta de liberación

BIBLIOGRAFÍA

Tema:
Calidad en la planeación de proyectos de desarrollo

Introducción

El presente trabajo de tesis tiene como objetivo principal, el planear y controlar cualquier proyecto de desarrollo, pues solo a través de técnicas y métodos bien fundamentados, se pueden llevar a cabo aplicaciones controladas, con un máximo aprovechamiento del grupo de programación y con la calidad requerida en todo proyecto.

La Ingeniería de Software será la encargada de minimizar los esfuerzos y de maximizar los resultados, mientras que una buena planeación, nos dará un mejor rumbo y un mayor control del proyecto, por lo que la combinación de ambas nos llevará a conseguir una mayor calidad en nuestros desarrollos, y nos brindarán la confianza de continuar aplicando las metodologías
hacia proyectos posteriores.

El capítulo uno se enfoca de manera general en los conceptos básicos de los lenguajes de programación, el cual sirve como referencia para conocer las estructuras básicas de control, los tipos de lenguajes y el desarrollo de ellos. Hace mención especial a los lenguajes de cuarta generación como precursores para que aquellas personas que desconocen de programación
y que desean realizar sus primeras aplicaciones. Menciona a los sistemas expertos y a la Programación orientada a objetos como posibles panacéas en el futuro de la programación. Este capítulo es una guía de referencia general para aquellos que se inician en el ámbito computacional.

En el capítulo dos se enfatiza la importancia de la ingeniaría de software para crear sistemas con alta calidad y con un bajo costo, además un sistema de software actual sería de difícil corrección ó mantenimiento, pues a través de sus controles que imperan desde el inicio del desarrollo es fácil distinguir cualquier cambio a realizar. Se presentan muestras representativas del alto costo del software y hardware y de ahí surge la importancia de implantar mecanismos que

garanticen la calidad de los desarrollos, es aquí en donde se profundiza en las pruebas de integridad y validación para cada desarrollo, pues ellas son la medida más importante del trabajo desarrollado. Durante el capítulo se recomiendan 5 documentos funcionales que sirven como pilares para llevar a cabo cualquier sistema de software.

El último capítulo nos muestra como podemos a través de la estructura funcional del personal, de la comunicación del equipo de desarrollo y de algunas otras herramientas ir planeando y controlando las actividades de desarrollo. También se hace énfasis en la estimación de todo proyecto, pues de ella dependerán decisiones de si se continúa con el proyecto o si se
abandona. Debido a que el tiempo aveces significa el enemigo a vencer en la mayoría de los proyectos y es por esto y por la ansiedad de empezar, que no nos queremos detener en invertir un poco de tiempo en ponernos de acuerdo con la planeación. La planeación detallada de todo proyecto de programación es fundamental para conocer el tiempo más aproximado en la
entrega del producto, minimiza los costos, incrementa la calidad y también reduce los posibles mantenimientos. La planeación es uno de los puntos clave para el logro de buenos resultados, sin embargo muchos proyectos demeritan esta actividad debido a que no cuentan inicialmente con el tiempo, la información ni metas claras del proyecto, aveces no se conocen las verdaderas necesidades del cliente ni las restricciones del producto.

Toda aplicación de software, puede ser susceptible a todos los puntos que se tratan en los capítulos dos y tres, esto sin importar lo grande o pequeña que parezca la aplicación, pues los temas son tan versátiles, que pueden ser adaptados al desarrollo de todo trabajo de software, pudiéndose omitir aquellos que por la naturaleza del proyecto no se utilicen.

Capítulo
UNO

CONCEPTOS BÁSICOS Y LENGUAJES DE PROGRAMACIÓN

I.- INTRODUCCIÓN AL SOFTWARE

Aveces nos cuestionamos como un ser humano puede comunicarse con un monitor y una caja que contiene componentes electrónicos, circuitos integrados y cables; nos hemos preguntado como podemos comunicarnos con una máquina que solo recibe pulsaciones eléctricas y que internamente transforma en ceros y unos: el encargado que ha permitido realizar esa comunicación es el **software**, el cual ha superado el hueco que existia entre el hardware de la máquina y las instrucciones que el usuario le queria transmitir, pero el software que hoy conocemos ha evolucionado de tal manera que nos permite comunicarnos de una manera muy sencilla y amigable, que permite que un usuario inexperto pueda escribir novelas, que realice la presentación de su proyecto escolar, que pueda realizar el control de gastos de su negocio a través de una hoja de calculo. El software puede transformar a nuestro monitor como por arte de magia en un procesador de textos, en un divertido juego, en una aplicación contable, en un diseñador gráfico, en un instrumento musical, etc. Casi todo el software corresponde a una de tres categorías principales :

- Programas de traducción
- Aplicaciones de software
- Software de sistema

I.1.- Programas de traducción

Los programas traductores permiten a los programadores escribir código en algún lenguaje de programación, los cuales son traducidos en ceros y unos que es el lenguaje que utiliza la computadora; los traductores liberan al programador de escribir código en lenguaje de máquina, produciendo programas con mejor calidad y propenso a cometer menos errores. En los últimos veinte años los lenguajes de programación han tenido una evolución considerable, con lo cual facilitan la tarea de los programadores asumiendo mas sobre el trabajo detallado y laborioso ; No obstante el ingeniero de software se enfrenta a muchos detalles técnicos y lógicos, lo cual requiere una buena dosis de tiempo y esfuerzo mental.

I.2.- Aplicaciones de Software

Son un instrumento para que los usuarios no programadores puedan desarrollar su trabajo con mayor grado de productividad, ya que estas herramientas simulan o extienden las aplicaciones de la vida común, con lo que sin ellas seria difícil realizar. Existen paquetes de software integrado, el cual permite combinar varias aplicaciones para poder desarrollar un trabajo realmente profesional.

Casi cualquier empresa o individuo trabaja con alguno de estos paquetes comerciales de software :

- Procesadores de texto y publicación electrónica
- Hojas de cálculo y aplicaciones de procesamiento numérico
- Bases de datos para almacenar y manipular información
- Software de telecomunicaciones y redes
- Aplicaciones gráficas
- Multimedia e hipermedia

Como ejemplo los procesadores de texto nos ofrecen la ventaja de dar formato e insertar imágenes y tablas a nuestro texto. factor que seria muy difícil de añadir con una máquina de escribir (ver figura 1.1).

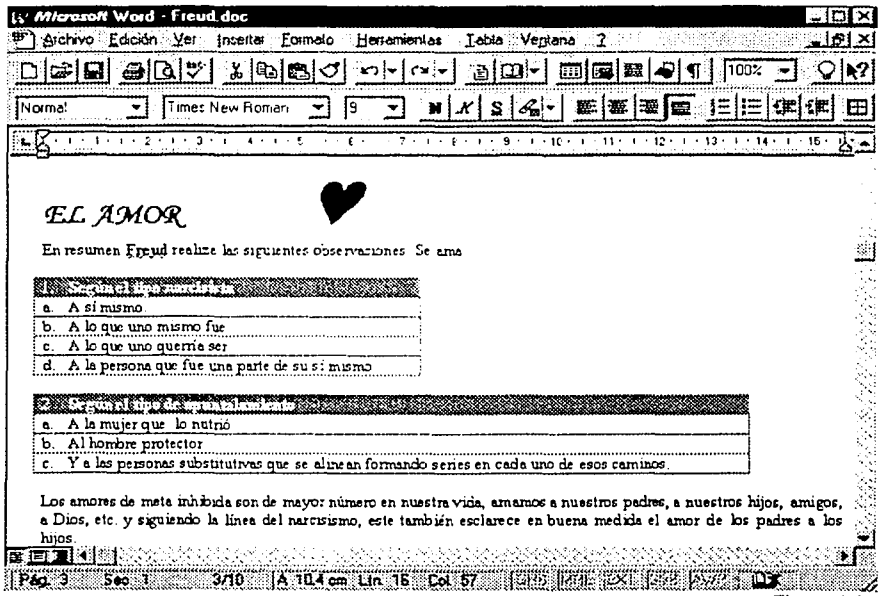


Figura 1.1

Estos paquetes contienen documentos impresos (manuales) que nos ayudan a entender el funcionamiento del software, así como ayuda en línea mediante pantallas complementarias de información, sin embargo estos paquetes son tan fáciles de comprender que muchas veces no es necesario tener que leer esta documentación.

El mundo de la industria del software incluye diferentes versiones para la actualización o corrección de errores en sus productos, así por ejemplo, para incluir nuevas características menores se incrementa en decimales la versión utilizada, pero si el software ha sufrido grandes modificaciones, entonces se incrementa al siguiente dígito, tal es el caso del paquete comercial "EXCEL" que se incremento de la versión 4.0 a la versión 5.0 con cambios muy importantes; Todo el software desarrollado esta protegido con los derechos de autor, de tal forma que no pueda ser duplicado ilegalmente, ni distribuido a terceras personas, también lo tratan de proteger físicamente contra copias pirata, debido a que la creación de software es sumamente costosa y de esta forma aseguran la recuperación económica del negocio.

I.3.- Software de sistema.

El encargado de administrar los recursos de la computadora y de ordenar la traducción del software para que el hardware lo pueda entender es el **Sistema Operativo (S.O.)**, el cual funciona tras bambalinas y sin que el usuario lo distinga, pero no por eso su papel deja de ser importante. El **S.O.** trabaja desde que encendemos la computadora, y realiza la misma función en una pequeña LAPTOP como en una MACROCOMPUTADORA de tiempo compartido, y sus funciones principales son :

- Comunicación con los periféricos. Algunos de los objetivos de los usuarios es presentar su trabajo mediante algún medio impreso o que la computadora le despliegue los resultados en pantalla, por lo que el sistema operativo se encarga de manera transparente de realizar la comunicación con estos y otros dispositivos periféricos.

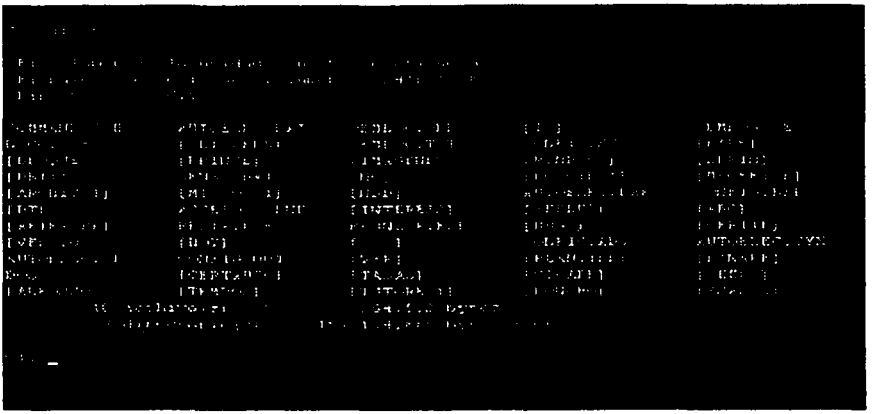
- Procesamiento de multitareas. Las actuales computadoras personales tienen la capacidad de realizar trabajos en forma simultánea, es decir aprovechan la inactividad de algún proceso (por ejemplo, la espera de una entrada) para dedicarle tiempo a otra aplicación, así mismo las macrocomputadoras multiusuario pasan de una terminal a otra verificando si hay procesos a realizar y corriendo simultáneamente otros, a esto también se le llama **procesamiento concurrente.**

- Administración de la memoria, programas y datos. Para el funcionamiento eficiente de los procesos concurrentes, el sistema operativo debe controlar los tiempos y recursos que le asigna a cada tarea a procesar, para asegurar que otra aplicación no se quede sin ser atendida. También nos ayuda a localizar archivos y darnos acceso a los mismos.

- Supervisión y seguridad de recursos. Con un sistema multiusuario podemos saber cuanto tiempo lleva trabajando una terminal, le podemos dar atributos de seguridad y acceso, así como asignarle un espacio determinado en disco, es decir, nos da el control de saber hasta que archivos esta utilizando el usuario, por lo que podemos explotar estas características para fines de registro e inspección.

A principios de los 80's la interfaz más conocida que utilizó el sistema operativo para comunicarse con los usuarios fue la comunicación en línea de ordenes de MS-DOS (Microsoft Disk Operating System : Sistema Operativo en disco Microsoft) basada en caracteres, que abarcaban una pantalla de 24 líneas por 80 columnas de texto, números y símbolos. Al iniciar MS-DOS nos aparece el prompt del sistema operativo y una pantalla vacía, La computadora espera recibir las instrucciones (comandos) en forma de ordenes y estas deben ser escritas correctamente, de lo contrario la computadora responderá con un mensaje de error.

Pantalla clásica de un sistema operativo basado en MS-DOS (Fig. 2.1).

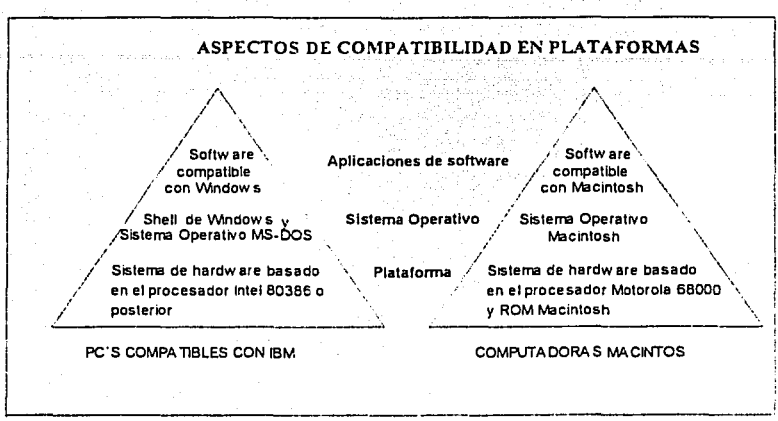


Interfaces gráficas con el usuario

El Sistema Operativo basado en su pantalla de caracteres MS-DOS obtuvo mucho éxito, sin embargo, esto representaba cierta dificultad en usuarios inexpertos, más aún cuando la computadora ocupo puestos importantes en industrias y oficinas, pues el personal no especializado no entendía o no se aprendía los comandos básicos para el manejo la computadora.

Tiempo después aparecieron las computadoras Macintosh (1984), los cuales fueron capaces de sustituir la memorización de comandos por el uso de pantallas totalmente amigables, las cuales permiten mediante el movimiento del ratón seleccionar alguna aplicación, o moverse a través de sus carpetas (directorios) para copiar o manipular información sin ninguna dificultad. El Sistema Operativo Macintosh se basa en una interfaz gráfica llamada **GUI** (graphic user interface) : la cual no se parece en nada a la línea de comandos del MS-DOS. Del mismo modo se desarrollo en otra plataforma[1] una interfaz capaz de esconder casi todas las características del MS-DOS, y que se pueden correr en máquinas basadas en computadoras personales compatibles con IBM, a este nuevo entorno se le llamó **ambiente Windows**.

[1] Plataforma: Representa al hardware en el cual podemos ejecutar nuestro software



Sin lugar a dudas, el desarrollo de las aplicaciones en Windows es una realidad, lo cual hace suponer que el futuro de la microprogramación va encaminado hacia los entornos gráficos. Una de las características principales que diferencia a Windows de DOS es un entorno gráfico que en base a ventanas presenta al usuario en forma de objetos la mayoría de operaciones que se pueden realizar con el sistema operativo, evitando así tener que recordar ordenes y opciones que en muchos casos el escribirlas ya era una posible fuente de errores. En un entorno gráfico el usuario simplemente selecciona la utilidad correspondiente a la tarea que desea ejecutar; el resto del trabajo lo hace el sistema.

Una de las grandes ventajas de trabajar con Windows es que todas las ventanas trabajan de la misma forma y todas las aplicaciones utilizan los mismos métodos básicos (menús desplegables, botones, etc.) para introducir ordenes.

Una ventana típica de Windows tiene las siguientes partes:
1.- Barra de título: Contiene el nombre de la ventana y el documento
2.- Barra de menús. Visualiza el conjunto de los menús disponibles para esa aplicación.- Cuando una de los menús se activa haciendo clic con el ratón sobre su título, se visualiza el contenido de ordenes que lo forman.
3.- Menú de control. Proporciona ordenes para restaurar a su tamaño, mover, minimizar, maximizar y cerrar pantalla.
4.- Barras de desplazamiento horizontal y vertical
5.- Área de trabajo. Es el área en donde la aplicación Windows reside.
Como ejemplo vease la figura 1.3.

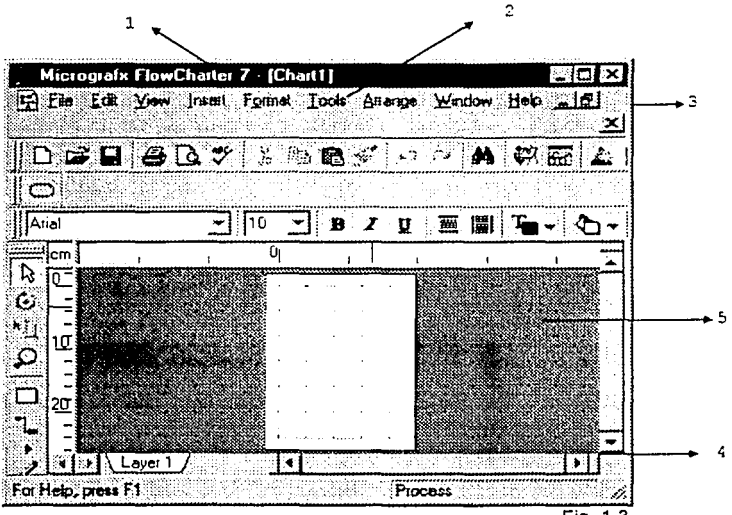
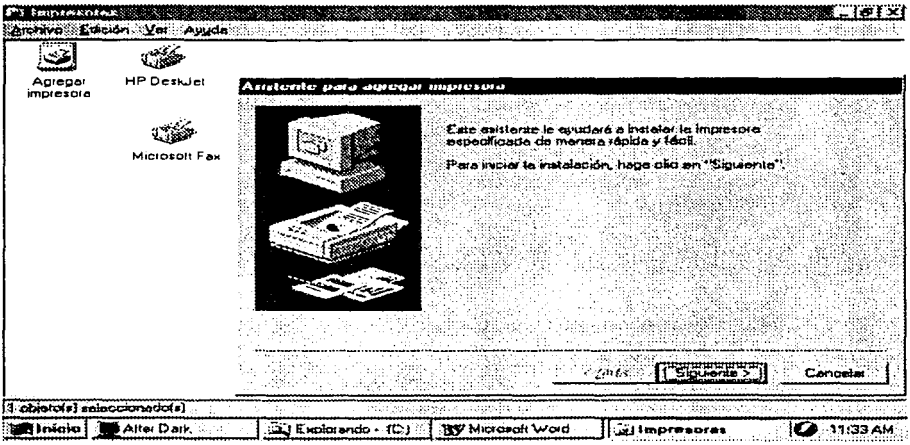


Fig. 1.3

En la siguiente ilustración (Fig. 1.4) podemos observar un asistente para la instalación de una impresora en el sistema operativo de Windows 95; este proceso hubiera resultado muy laborioso hace algunos años para usuarios no expertos, sin embargo con la ayuda de las interfaces gráficas resulta un proceso simple.



De hecho sabemos que todos los usuarios prefieren las interfaces gráficas, y que cada vez se ocultan más las ordenes del MS-DOS, sin embargo cual será la tendencia de los Sistemas Operativos del futuro? Bueno, la tendencia marca que seguiremos con las interfaces gráficas por un tiempo considerable, sin embargo en laboratorios de investigación se usa el lenguaje natural, para que una computadora reconozca un subconjunto del idioma y así pueda ejecutar cierta orden, estas capacidades también comienzan a verse en el mercado, las máquinas del futuro serán capaces de manejar gran parte del trabajo cotidiano a través de una interfaz en lenguaje natural, ya sea hablado o escrito, así mismo terminarán detalles que se nos pasen por alto, anticiparán actividades que cotidianamente realizamos y organizarán de forma automática el espacio de nuestra máquina de acuerdo a nuestras necesidades.

Otros Sistemas Operativos Gráficos:

- OS/2, de IBM es un sistema operativo de alto poder que puede ejecutar aplicaciones creadas para DOS y Windows ; es comúnmente utilizado en industrias y empresas.
- Microsoft Windows NT, es un sistema operativo completo basado en Windows y diseñado para operar en computadoras personales de alto nivel y estaciones de trabajo ;
- New Wave DE Hewlett Packard, un shell que se monta sobre un shell Windows y cambia el interfaz con el usuario para que parezca más al de Macistosh ;
- NextStep, de Next, Inc ; es un poderoso y complejo sistema operativo montado sobre UNIX

II.- CONCEPTOS DE PROGRAMACIÓN

De manera muy general un programa se define como una serie de acciones o instrucciones a seguir, por lo tanto una receta de cocina, una partitura musical, o las instrucciones para armar el modelo de avión a escala son todos programas; sin embargo de acuerdo a nuestro contexto un programa se define como una ***"serie de instrucciones o proposiciones" capaces de ser procesadas y ejecutadas de manera ordenada por una computadora.***

Un programa también es visto como una caja negra, en donde se reciben los datos de entrada, éstos son procesados y se obtienen a la salida los resultados esperados.

Por lo tanto vemos que un programa es esencialmente un medio de comunicación que utiliza el hombre para decirle a la máquina lo que debe de hacer y como debe de hacerlo.

Un programa tradicional comprende:

- Un principio que se realiza cuando se ejecuta el programa principal y comprende normalmente las puestas en estado inicial de la memoria central, que pueden consistir en establecer y declarar constantes, variables, asignación de arreglos, etc. también podemos definir colores de entrada (presentación),determinar comentarios iniciales, numero de decimales a manejar, etc.

  La puesta inicial de un programa realizado en CLIPPER podría verse de la siguiente forma:

```
*****************
* MENU.PRG      *
*MENU PRINCIPAL *
*****************
A).- SET DATE BRITISH
B).- SET CURSOR OFF
C).- SET BELL ON
D).- XRESP:=SPACE(1)
E).- XDIA:=0
F).- XFECHA:=CTOD(" / / ")
G).- SET COLOR TO W+/BG,N/W
```

  La primera instrucción se encargará de definir nuestro formato de fechas en forma británica "DD/MM/AA", la segunda instrucción hará que no se pueda visualizar el cursor dentro de nuestro programa, hasta que éste sea activado, la instrucción "C" emite un pitido cada vez que se rellene un campo a ser leído, las demás instrucciones son para inicializar nuestras variables de propgrama, así como para determinar el color que se desplegará al iniciar nuestra pantalla.

- El **cuerpo del programa** comprende normalmente la parte principal de nuestro código y esta compuesto a su vez por un conjunto de instrucciones, las cuales pueden llegar a ser ejecutadas un sin numero de veces; por ejemplo si queremos dar de alta a 100 empleados dentro del sistema de nomina de alguna empresa, el proceso se tendrá que repetir tantas veces como se lo indiquemos a la máquina, es decir, después de cada alta el sistema preguntará si se desea dar de alta a otro empleado, si la respuesta en positiva se volverá a ejecutar el bloque de código utilizado anteriormente.

  A su vez el cuerpo del programa comprende: la lectura de datos (la cual se realiza a través del teclado, bases de datos, unidades de cinta magnética, etc.), la ejecución de cálculos y comparaciones, la preparación de los datos e información a la salida y la impresión o la escritura de los resultados.

- Fin del programa. Cuando el cuerpo del programa se ha terminado de utilizar, al final se utilizan una serie de instrucciones que se ejecutan una sola vez, algunas de estas instrucciones podrían ser: cierre de bases de datos, regreso de los colores originales de la pantalla, activar funciones del cursor, regresar al sistema operativo de donde fue invocado, etc.

III.- COMPONENTES DE UN PROGRAMA

a su vez un programa se compone de **estructuras de datos, operaciones elementales y estructuras de control**, que a continuación se describen:

III.1.- Estructuras de datos

Una representación de hechos conceptos o instrucciones de una manera formal, apropiada para la comunicación, la interpretación o el procesamiento se denomina como una estructura de datos, es decir, los hechos reales representados en forma de datos, pueden estar organizados de diferentes maneras llamadas estructuras de datos. Estas estructuras permiten el manejo de grandes volúmenes de datos mediante operaciones relativamente sencillas. las estructuras de datos mas importantes que se emplean son cadenas, colas, pilas, matrices, listas y arboles; y para el almacenamiento masivo de datos se emplean: ficheros, registros y campos.

Un ejemplo de una estructura de datos podrían ser las palabras que utilizamos en nuestro lenguaje que se enlazan en frases y oraciones para poder construir un lenguaje y así podemos comunicar. Otros ejemplos de información estructurada son los diccionarios, la sección amarilla, las enciclopedias, etc. Todas estas colecciones de información no nos servirían si no estuvieran organizadas de acuerdo a unas cuantas reglas, estas estructuras organizadas nos permiten el acceso rápido y fácil a un dato en particular.

III.2.- Operaciones elementales

Son las acciones que se ejecutan sobre los datos para poder obtener algún resultado, estas operaciones pueden ser: suma, resta, multiplicación, división, comparación, transferencia, etc. y todas ellas tienen un determinado orden de prioridad o precedencia que especifica la forma en que se ejecutarán las operaciones, este orden de operación en la mayoría de los lenguajes de programación es el siguiente:

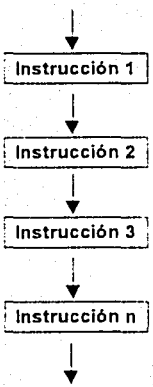
1.- Paréntesis
2.- Potencia
3.- Producto y división
4.- Sumas y Restas
5.- concatenación
6.- Relacionales
7.- Negación
8.- Conjunción
9.- Disyunción

Las operaciones que tienen la misma prioridad se realizan de izquierda a derecha.

III.3.- Estructuras de Control

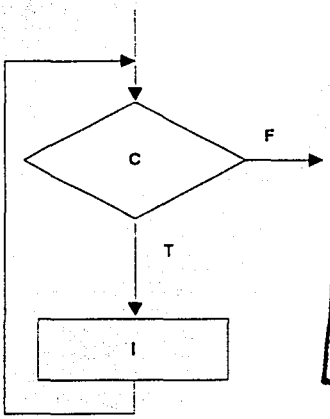
Las estructuras de control son los métodos que se disponen para transferir el control de una parte a otra sobre los datos que se manejan en el programa. Básicamente se utilizan tres estructuras de control que son: el secuenciado, los bucles y las bifurcaciones.

El **secuenciado** es el flujo que se sigue cuando se va de una instrucción a otra, es decir, se consigue cuando la operación se transfiere a la inmediata siguiente:

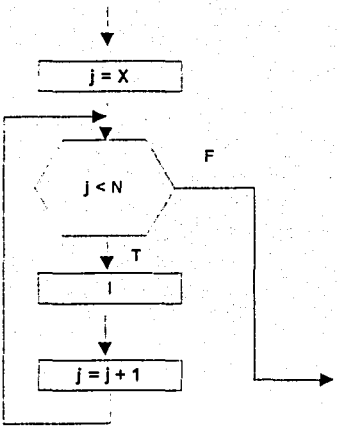


Las instrucciones de **repetición o bucles**, se utilizan para realizar varias veces una acción, de acuerdo a que cierta condición sea falsa o verdadera o de acuerdo a una variable utilizada como contador.

Do while:
While C do I

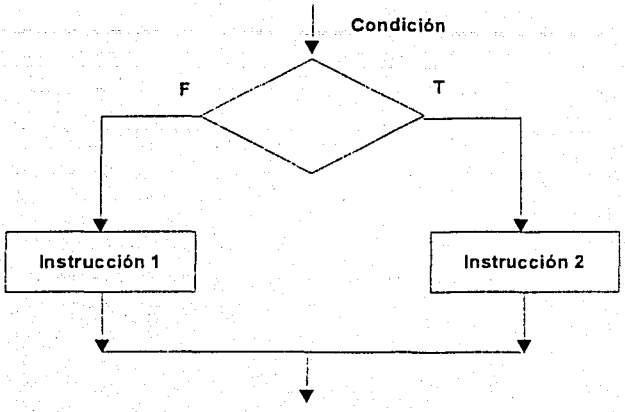


```
For j = X to N
    I
Next
```

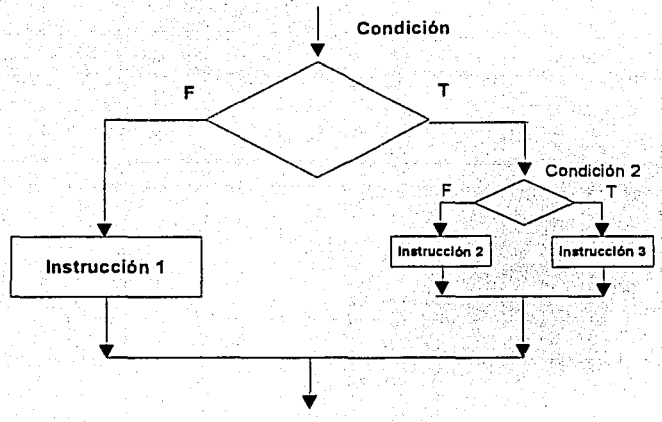


Las instrucciones de bifurcación o selección (si, entonces) permiten transferir el control de un lugar a otro dentro de un programa o a otros programas, solo si la condición es verdadera y opcionalmente se ejecutará la otra parte en caso de que exista una condición falsa. Dentro de una misma instrucción de selección pueden existir otras más, a esto se le conoce como anidamientos; dentro de este tipo de anidamiento puede existir cierta confusión para poder entender la lógica del programa, por lo que se creó una instrucción mas sencilla de manejar llamada "bifurcación en sentido múltiple" la cual en muchos lenguajes de programación se le conoce como "Case".

Ejemplo de bifurcación simple:



Ejemplo de bifurcación con anidamiento:



IV.- PERSONAL LIGADO A LA VIDA DE UN PROGRAMA

Dentro de una empresa la vida de un programa está ligada a cierto grupo de personas, las cuales conviven con el sistema en diferentes etapas del desarrollo del proyecto.

IV.1.- Analista de Sistemas

Las primeras personas que ven nacer la posibilidad de un nuevo proyecto son los analistas de sistemas, los cuales deben poseer las cualidades siguientes:

a).- Conocimientos generales del departamento y área, así como sus objetivos a corto, mediano y largo plazo y de sus actividades físicas o económicas a las que se dedica.
b).- Conocimientos prácticos de administración y contabilidad.
c).- Conocimientos sobre las prácticas de procesamiento de datos, así como las diferentes técnicas de programación.
d).- Conocimientos amplios en diferentes lenguajes de programación.
e).- Conocimientos generales sobre el hardware de las computadoras.
f).- Un alto nivel en su creatividad incluyendo la capacidad de transmitir ideas en forma clara y precisa; así como el trabajar de manera cooperativa y amable con el diferente personal que colabora con él.

Esta última característica también es importante, debido a que las buenas relaciones que pueda llevar con el demás personal(gerentes, programadores, operadores, etc.) conduce a una mejor comunicación y por lo tanto a un sistema más real.

Los analistas de sistemas deben realizar mejoras a los sistemas existentes y desarrollar ideas innovadoras para futuros proyectos, al personal que cumple con éstas características y con algunas otras mencionadas anteriormente, comúnmente se les encargan responsabilidades de dirigir un proyecto.

El analista debe llevar una comunicación estrecha con los usuarios finales para determinar sus requerimientos o necesidades y así poder empezar a convivir con el sistema desde el análisis, el diseño, desarrollo y liberación.

IV.2.- Programadores

El 2do. grupo de personas que convive con el sistema es el personal de programación, el cual debe tener las siguientes características:

a).- Debe conocer ampliamente la sintaxis y las estructuras de control de los lenguajes que se utilizaran en el proyecto así como entender y poder aplicar las técnicas generales de programación.
b).- Debe tener una creatividad lógica y disciplinada en su estilo de programación así como la capacidad de entender fácilmente lo que se le explica y lo que se quiere obtener como resultado.
c).- Conocer las relaciones existentes entre el software y la(s) máquinas en donde se instalará el sistema, ya que en algunos casos el programa no podrá correr correctamente, debido a las características de hardware de cada equipo.
d).- Deben ser pacientes y detallistas, ya que algunos programas se componen de cientos de líneas de código (Aveces miles o millones) , por lo que resulta difícil y tedioso rastrear algún error lógico.
Si el programador realiza un buen trabajo desde el principio, éste se verá favorecido a la hora de realizar cualquier cambio dentro del programa, es decir, el mantenimiento resultará factible mediante una buena documentación y una buena lógica de programación.

Un cambio pequeño en alguna parte de un programa grande puede dar como resultado alguna falla secundaria, por lo que la evaluación y corrección cuidadosa de errores, son sin duda un trabajo para gente con mucha paciencia y sumamente detallistas.

IV.3.- Usuarios

Los usuarios finales son el otro tipo de personas que conviven con el sistema, estos se encargan de ingresar y/o manipular información a través del programa para poder obtener los resultados esperados; ellos ya no tienen que meterse con el funcionamiento interno de la aplicación, pero si se les requiere para obtener información necesaria y así poder retroalimentar y realizar mejoras al sistema ; las características que son deseables en este tipo de personal son:

a).- Deberá tener conocimientos básicos acerca del funcionamiento de la computadora, (encendido, manejo de teclado, mouse, etc).
b).- Debe conocer a grandes rasgos el funcionamiento del programa.
c).- Deberá tener la iniciativa para aportar ideas de como mejorar el sistema.
d).- Todos los usuarios que participen en la entrada de datos, deberán adiestrarse con el fin de reducir el índice de errores en la captura de los datos.

V.- HERRAMIENTAS BÁSICAS DEL PROGRAMADOR

Se puede pensar que la herramienta mas indispensable para que una persona pueda desarrollar un programa es una computadora, sin embargo esto implica tener dentro de nuestra máquina lo siguiente :

a).- Para poder ingresar nuestras líneas de código se necesita introducirlas a través de un **editor de textos** el cual es similar a un procesador de palabras comercial, pero sin tener algunas características de formato, éste también nos servirá para añadir mas líneas o para realizar los cambios que se consideren necesarios. Existen editores de textos especiales para escribir programas de todo tipo de lenguaje, es decir se graban como caracteres ASCII y posteriormente se compilan o interpretan de acuerdo al lenguaje que estemos utilizando; también existen editores de textos interactivos. Algunos son diseñados ofreciendo sangrías automáticas dependiendo de las estructuras de control a utilizar y algunos otros ofrecen una pequeña revisión de errores mientras se teclea el programa.

b).- Una vez escrito nuestro código fuente, éste debe ser filtrado para saber si fue escrito sin errores de semántica o de sintaxis, por lo que también el programador deberá tener contar con un **compilador o intérprete**.

c).- **Un sistema manejador de archivos ó administrador de proyectos**, el cual nos permita administrar nuestros archivos, y a través de él tener un mejor control sobre los archivos de mayor importancia, por ejemplo, podríamos restringir el acceso a aquellas personas que quisieran editar nuestros archivos fuente, o que quisieran copiarlos o correrlos. También nos permite conservar las distintas versiones de nuestros programas fuente y sus inter - relaciones.

d).- También es recomendable tener siempre a la mano un **manual de referencia** en donde el programador pueda consultar rápidamente sus dudas acerca del lenguaje de programación que esté utilizando, afortunadamente casi todos los lenguajes ya poseen una **guía en línea** de todo lo referente a los mandatos, funciones, comandos, estructuras de control, etc.

e).- También existen **paquetes de programación** muy completos, los cuales son herramientas de programación integrados que ya contienen un editor de textos, un compilador, un administrador de proyectos, ayuda en línea y un debbuger para facilitar el proceso de corrección de errores.

VI.- LENGUAJES DE PROGRAMACIÓN

Actualmente se emplea software necesario para resolver muchos de nuestros problemas, tareas o trabajos. Este software de aplicación esta hecho por profesionales de la computación quienes trabajan para casas comerciales de la informática, sin embargo muchos de nuestros problemas o trabajos no solo requieren de estos "paquetes de computación". Los lenguajes de programación no son aplicaciones, si no herramientas que nos permiten construir nuestras propias aplicaciones y adecuarlas para satisfacer nuestras necesidades mas especificas.

VI.1- Niveles de lenguajes de programación

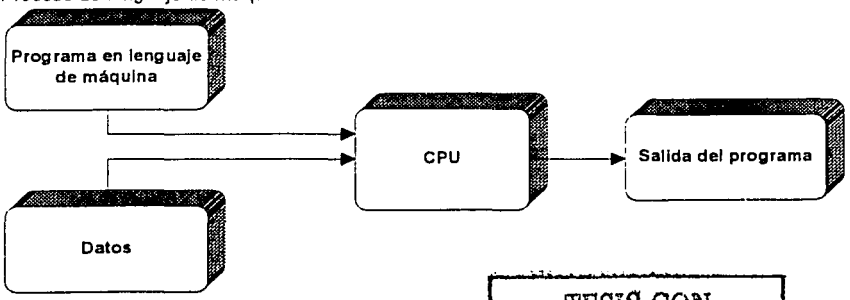
Como se dijo anteriormente, un programa de computadora consiste en un conjunto de instrucciones a realizar para conseguir una tarea específica. Los programas se pueden escribir en distintos niveles de lenguajes de programación, estos pueden ser: **lenguaje de máquina, lenguaje ensamblador y lenguaje de alto nivel.** Sin embargo los programas escritos en lenguaje ensamblador o de alto nivel (programas fuente) finalmente deben ser traducidos en lenguaje de máquina (programa objeto), debido a que la computadora internamente solo entiende este tipo de lenguaje (ceros y unos). Para entender mas a fondo lo que son estos tipos de lenguajes, a continuación se da una explicación de lo que representa cada uno de ellos.

VI.1.1- Lenguaje de máquina

El lenguaje de máquina surge en la década de los 40's como resultado de la invención de las primeras computadoras en esas épocas, éste lenguaje consiste en sucesiones de dígitos binarios (1's y 0's) que permiten programar todas las instrucciones o mandatos valiéndose de estas cadenas de dígitos. Programar en lenguaje de máquina hace muy eficiente el aprovechamiento de los recursos de la computadora debido a que el programa interactúa directamente con la Unidad de Procesamiento Central; sin embargo la programación en este tipo de lenguaje requiere un conocimiento más profundo de la máquina, así como conocer las características especiales de la computadora, otra desventaja en este nivel de programación es la necesidad de conocer todos los códigos de operación existentes para el tipo de máquina que se vaya a utilizar.

Aun hoy en día, el único lenguaje que entiende la computadora es el lenguaje de máquina, ya sea una microcomputadora o una macrocomputadora, sin embargo los programadores actuales ya no programan en este tipo de lenguaje debido a lo laborioso que resulta, mas bien lo hacen en lenguajes de alto nivel, que mas tarde son traducidos en lenguaje de máquina a través de traductores(compiladores e interpretes).

Proceso de lenguaje de máquina



VI.1.2- Lenguaje ensamblador

Los lenguajes ensambladores se originaron a finales de los años 50´s como una alternativa para superar el entendimiento de los lenguajes de máquina. Un ensamblador es un programa escrito en lenguaje de máquina que permite a los programadores escribir en un lenguaje más fácil de entender. Los ensambladores antiguos tenían una correspondencia uno a uno con las instrucciones del lenguaje de máquina, pero en la actualidad existen macroinstrucciones que permiten al programador utilizar una sola seudoinstrucción que traduce más de una instrucción en lenguaje de máquina; este tipo de lenguaje es denominado "macroensamblador", las macroinstrucciones que contienen los códigos de operación y las direcciones del operando son llamadas "mnemónicos" los cuales son más fáciles de recordar y de programar.

Sin embargo el ensamblador aún es un lenguaje propenso a errores y muy tedioso, no obstante en la actualidad el lenguaje ensamblador aún es utilizado por varios programadores para escribir juegos de video o cuando lo que les interesa es un aprovechamiento del nivel máximo en la eficiencia de la máquina en cuanto a la velocidad y comunicación directa con el hardware.

COMANDOS MAS UTILIZADOS EN EL LENGUAJE DE ENSAMBLE :

NOMBRE	SÍMBOLO DE OP.	CÓDIGO	FORMATO	OPERANDOS
*Add(c)	AR	1A	RR	R1,R1
*AND(c)	NR	14	RR	R1,R2
*Brach and Link	BALR	05	RR	R1,R2
*Brach on index low or equal	BXLE	87	RS	R1,R3,D2(B2)
*Clear Channel (c,p)	CLRCH	9F01	S	D2(B2)
*Compare(c)	CR	19	RR	R1,R2
*Compare and Swap(c)	CS	BA	RS	R1,R3,D2(B2)
*Compare Logical(c)	CLR	15	RR	R1,R2
*Convert to Binary	CVB	4F	RX	R1,D2(X2,B2)
*Convert to decimal	DVD	4E	RX	R1,D2(X2,B2)
*Divide	DR	1D	RR	R1,R2
*Edit(c)	ED	DE	SS	D1(L,B1),D2(B2)
*Edit and Mark(c)	EDMK	DF	SS	D1(L,B1),D2(B2)
*Exclusive OR(c)	XR	17	RR	R1,R2
*Execute	EX	44	RX	R1,D2(X2,B2)
*Extract PrimaryASN(s)	EPAR	B226	RRE	R1
*Halt Device(c,p)	HDV	9E01	S	D2(B2)
*Insert Character	IC	43	RX	R1,D2(X2,B2)
*Load	LR	18	RR	R1,R2
*Load Address	LA	41	RX	R1,D2(X2,B2)
*Load Complement(c)	LCR	13	RR	R1,R2
*Load Control(p)	LCTL	B7	RS	R1,R3,D2,(B2)
*Load Negative(c)	LRN	11	RR	R1,R2
*Load Positive(c)	LPR	10	RR	R1,R2
*Monitor Call	MC	AF	SI	D1(B1),12
*Move	MVI	92	SI	D1(B1),12
*Move Inverse	MVCIN	E8	SS	D1(L,B1),D2(B2)
*Multiply	MR	1C	RR	R1,R2
*OR(c)	OR	16	RR	R1,R2
*Pack	PACK	F2	SS	D1(L1,B1)D2(L2,B2)
*Program Calls(c)	PC	B218	S	D2(B2)

*Program Transfer(s)	PT	B228	RRE	R1,R2
*Purge TBL(p)	PTLB	B20D	S	
*Read Direct(p)	RDD	85	SI	D1(B1),12
*Reset Reference Bit(c,p)	RRB	B213	S	D2(B2)
*Set Clock(c,p)	SCK	B204	S	D2(B2)
*SET CPU Timer(p)	SPT	B20B	S	D2(B2)
*Set Storage Key(p)	SSK	08	RR	R1,R2
*Shift And Round Decimal(c)	SRP	FO	SS	D1(L1,B1)D2(L2,B2),13

C Se determina el código de condición
n Se carga un nuevo código de condición
p Instrucción privilegiada
s Instrucción semiprivilegiada
x Punto flotante, precisión extendida

VI.1.3- LENGUAJES DE ALTO NIVEL

La otra posibilidad de escribir un programa es en los lenguajes de alto nivel, estos se desarrollaron ampliamente al inicio de los años 60's para facilitar el entendimiento entre la computadora y el programador, ya que las instrucciones son más fáciles de comprender que las del lenguaje ensamblador y el programador no tiene que saber acerca de las características del hardware de su computadora, aunque siempre debe obedecer a una sintaxis rígida para cada lenguaje que trate.

Este tipo de lenguajes se asemejan a lo que es el lenguaje natural (EN INGLES) , para después ser traducidos a lenguaje de máquina (programa objeto) y así la computadora lo pueda procesar. Los lenguajes de nivel superior aparte de facilitar la tarea de escribir un programa, alientan a los programadores a buscar mejores soluciones para los problemas y aceleran el desarrollo de los mismos, debido a que un lenguaje de alto nivel simplifica todos los detalles e instrucciones laboriosas que realiza la máquina. Además Tiene la ventaja de poderse instalar en casi todas las máquinas existentes. Por ejemplo un programa escrito en lenguaje "C" puede compilarse y ejecutarse en cualquier tipo de máquina y rara vez se tendrá que reescribir para ajustarse a las diferencias del hardware.

Los objetivos principales de los lenguajes da alto nivel son el de acelerar el desarrollo de las aplicaciones que requieran programación con lenguajes y que el número de personas que los utilice se incremente cada vez más.

Así mismo tratan de resolver un determinado tipo de problema, no importando la máquina en la cual se trabaje, esto lo hace a través de una combinación entre el lenguaje natural y el lenguaje matemático, para poder dar forma así a lo que llamamos lenguajes de alto nivel; otro de los objetivos de los lenguajes de alto nivel es el de obtener la mayor **simplicidad** para la solución de problemas, ya que lo fácil debe ser fácil y lo difícil debe ser realizable. Finalmente un lenguaje de alto nivel persigue el aprovechamiento máximo de la eficiencia en sus procesos, así como tener una clara legibilidad en su código.

Limitaciones de los lenguajes de alto nivel :

A pesar de que los lenguajes de alto nivel representan una mejora notoria respecto a sus antecesores, aún siguen siendo incapaces de satisfacer necesidades mas profundas de los programadores:

- Los lenguajes de alto nivel todavía están muy cerca de lo que es la forma de trabajar de la computadora, ya que se tienen que especificar todos los pasos para la solución de un determinado proceso.

- La corrección de errores aún sigue siendo muy tediosa en programas medianos, y todo un reto en programas con miles de líneas, ya que se pueden generar nuevos errores al corregir otros.

- La sintaxis para escribir un programa sigue siendo muy rígida, por lo que aún se pierde mucho tiempo en tener el mayor cuidado al escribir un programa.

- Todavía en algunos lenguajes se necesitan varias decenas de líneas de código para solamente dibujar una pantalla de presentación.

Caracteristicas que definen a los lenguajes de alto nivel:

• Palabras reservadas y juego de caracteres
Las palabras reservadas son instrucciones especiales del propio lenguaje, y que no deben ser utilizadas como variables o constantes dentro de nuestro programa, así por ejemplo la instrucción READ o WRITE es utilizada para leer y escribir en muchos lenguajes, por lo que deben ser utilizadas específicamente para ese fin.
Los caracteres que se pueden emplear en un lenguaje son la colección de caracteres ASCII que puedan utilizarse en el programa. Considerando a un carácter como la parte mas diminuta en el tratamiento de la información. Los caracteres se pueden dividir en::
Caracteres alfabéticos: Los cuales van de la A ala Z, en minúsculas y/o mayúsculas
Caracteres numéricos: (0,1,2,...9).
Caracteres especiales: (.,-,*,&,|, etc.)

• Facilidades en la estructuración de programas
Los lenguajes de alto nivel poseen diversas formas de estructurar su código y de hacerlo más entendible, esto lo consiguen mediante subrutinas, procedimientos y funciones, los cuales permiten al programador llevar un mejor control en todos sus procesos, así como el de eficientar su código ; por ejemplo si en un almacén se necesita calcular el promedio neto de ventas de un determinado artículo, entonces se mandará llamar a éste proceso tantas veces como sea necesario.
Todos los lenguajes de todos los niveles nos presentan métodos para transferir el control de un lugar a otro, pero solo los lenguajes de alto nivel nos permiten la posibilidad de utilizar las 3 estructuras básicas de control, con las cuales se puede realizar cualquier tipo de programa por mas complejo que sea

• Datos y tipos de datos
Los datos que utiliza un programa pueden ser constantes o variables y las variables pueden ser a su vez públicas o privadas, lo cual nos da una visión del alcance que puede tener una variable dentro de nuestro programa. La mayoría de los lenguajes de alto nivel requieren que se declaren todas sus variables, ya que si no lo hacen causará un error a la hora de leer un dato o de escribir un resultado.

Los tipos de datos mas utilizados en los lenguajes de alto nivel son los numéricos y los alfanuméricos, los cuales pueden ser números enteros y reales y los alfanuméricos están

compuestos por cualquier tipo de carácter(alfabético, numérico, especial). Otro tipo de datos muy utilizados son los booleanos o lógicos, éstos pueden tomar los valores de falso o verdadero para ser utilizados como banderas en los programas. Los punteros son otro tipo de dato que brindan algunos lenguajes de alto nivel, estos son datos que contienen la dirección de otro dato y son muy útiles para la construcción de listas o arboles , sin embargo si llegara a existir un error, éste seria muy difícil de detectar.

• Operadores

Todos los lenguajes de alto nivel cuentan con operadores que nos permiten manipular nuestros datos de una forma sencilla y práctica. Estos pueden englobar en una sola sentencia varias instrucciones complejas mediante el uso de paréntesis. Existen algunas diferencias mínimas en la manera de asignar o comparar expresiones, de lenguaje a lenguaje, pero las más comunes son las siguientes :

Operadores aritméticos: (^) potenciación,(*) producto,(/) división,(+) suma, (-) resta
Operadores alfanuméricos: (+, &) concatenación
Operadores relacionales: (=) igual,(<) menor, (>) mayor,(>=) mayor o igual,(<=) menor o igual
(<>) diferente

operadores lógicos : (NOT) Negación, (AND) Conjunción, (OR) Disyunción

Tablas de verdad de los operadores lógicos:

OPERADOR NOT

X	NOT X
F	T
T	F

OPERADOR AND

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

OPERADOR OR

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

VII.- DESARROLLO DE LOS LENGUAJES DE ALTO NIVEL

El primer lenguaje de alto nivel que apareció en forma comercial fue el FORTRAN que significa (Formula Translation : Traducción de fórmulas) fue diseñado por John backus de la empresa IBM en 1953, el cual fue diseñado para resolver problemas científicos y de ingeniería. Este lenguaje representaba el comienzo de una nueva era de programación y en donde los costos de tiempo y esfuerzo se ven reducidos. Actualmente FORTRAN es utilizado en muchas universidades y por muchos científicos e ingenieros, quienes utilizan una versión mas actualizada.

Sin embargo FORTRAN siendo un lenguaje orientado para los científicos y teniendo un enfoque meramente matemático, no representa un estilo apropiado para generar aplicaciones empresariales es por ello que el gobierno de los estados unidos solicitó el diseño de un nuevo lenguaje para la administración de sus negocios, el resultado fue COBOL (Common Business Oriented Language: Lenguaje Común Orientado a Empresas) quien favoreció ampliamente a las aplicaciones comerciales y es ujtilizado en un amplio mundo de programadores dedicados ala programación para empresas en macrocomputadores.

El lenguaje de programación mas conocido por los estudiantes principiantes y mas difundido en el mundo es BASIC (Beginner's All-purpose Symbolic Instruction Code: Código de instrucciones simbólicas de propósito general para principiantes) El cual fue desarrollado a mediados de la década de 1960 por los profesores John kemeny y thomas kurtz de la universidad de DARTMOUTHH, como una alternativa de programación distinta a las actuales, debido a que si se deseaba diseñar un programa en FORTRAN o COBOL, éste se tenía que entregar en tarjetas perforadas y esperar varias horas para obtener los resultados de la compilación, en caso de que existiera algún error se tendría que repetir el proceso hasta que quedara libre de errores. BASIC permitió corregir estos problemas mediante una traducción del programa objeto mediante un intérprete, el cual permite la corrección instantánea de los errores.

Muy pronto BASIC alcanzó gran popularidad por su facilidad de manejo y su lenguaje sencillo, sin embargo frecuentemente presentaba líneas confusas, en donde el programador se podía perder fácilmente debido al uso constante de la instrucción "GOTO" en donde se envía el control del programa a otra ramificación del mismo. Esto era un verdadero problema en programas de mayor tamaño, debido a que se incrementaba el camino a seguir, ocasionando el uso excesivo de la instrucción GOTO.

Las técnicas de la programación estructurada hicieron su aparición para resolver este problema con la inclusión del lenguaje PASCAL, a principios de la década de los 70's, hecho por el científico suizo Niklaus Wirth (llamado así en honor de Blaise Pascal). La técnica que utiliza este lenguaje es el de no depender de la instrucción GOTO para controlar el flujo de control, si no más bien se basa en programas mas pequeños llamados módulos, que a su vez se componen de otros submódulos que se combinan mediante las tres estructuras básicas de control que son :el secuenciado, la repetición o bucle y la bifurcación.

Por todo lo que representa PASCAL y con la inclusión de la programación estructurada, éste se impuso como unos de los lenguajes más utilizados en al ámbito educativo de todo el mundo, y marco la pauta para que los programadores fijaran sus técnicas a estandarizar en la programación estructurada.

PASCAL constituyó un gran cambio en las técnicas de programación, sin embargo su manera de interactuar con el aprovechamiento máximo de la máquina está muy lejos de lo deseable; generalmente las aplicaciones no requieren aprovechar al 100% los recursos ni el tiempo de ejecución de la computadora, pero cuando existen aplicaciones en donde la velocidad y la conservación de la memoria son indispensables, entonces el programador necesitará mucho más que un buen lenguaje estructurado.

Para solucionar los problemas de eficiencia y rapidez surge a principios de los años 70's el lenguaje "C" creado en los LABORATORIOS BELL, el cual ofrece todas las características de la programación estructurada y además si se maneja con cuidado puede llegar a ser tan rápido y eficiente como el lenguaje ensamblador. Este tipo de lenguaje es muy completo y podemos hacer con el lo que la imaginación nos disponga, sin embargo para los estudiantes es un poco mas difícil de aprender que sus antecesores, aún así debido a su poder y eficiencia es un lenguaje utilizado ampliamente por los profesionales de la computación. El poderoso lenguaje "C" es utilizado como herramienta para crear sistemas operativos (UNIX) o para crear otros lenguajes de programación.

Desde esas fechas existen cientos de lenguajes de programación, sin embargo solo unos cuantos son utilizados en las empresas, industrias, centrales nucleares, campos militares, universidades, etc.

Otros lenguajes conocidos:

- ALGOL (Algorithmic Languaje : Lenguaje Algoritmico) Lenguaje desarrollado entre 1957 y 1962, surge como una necesidad de hacer un lenguaje de carácter universal, que tuviera mas claridad en el manejo de sus estructuras de control y fuera más independiente de un modelo en particular de computadoras debido a que sus competidores no alcanzaban estos requisitos, sin embargo los demás lenguajes pronto se perfeccionaron y ALGOL paso a ser un lenguaje poco usual.
- ADA (Fue llamado así en honor de la sra. Ada Lovelace, quien fue precursora de la computación en el siglo XIX) Lenguaje creado en a finales de los 70's para el uso del departamento de defensa de los estados unidos, sin embargo no ha pasado a ser parte comercial, sino meramente militar.
- LISP (List Procesing: Procesamiento de listas) Desarrollado por MIP a finales de los años cincuenta para procesamiento de datos no numéricos, tales como caracteres y otros símbolos. Este tipo de lenguaje se utiliza en la investigación de la inteligencia artificial, al igual que PROLOG (Programming Logic: Programación de lógica), quien está diseñado para trabajar con relaciones lógicas entre hechos.
- RPG (Report Program Generator: Programa Generador de reportes) Este tipo de lenguaje tuvo su origen a mediados de los años 60's y esta diseñado para facilitar el trabajo de obtener reportes e informes que no requieran muchos cálculos aritméticos, mucha gente principiante en la programación de aquella época lo entendía mas fácilmente que a sus oponentes, debido a que tiene un formato más flexible de escritura y posee muchas facilidades que los otros lenguajes carecían, como es el manejo de la instrucción calculate.

VIII.- TIPOS DE LENGUAJES DE ALTO NIVEL

VIII.1- Lenguajes de procedimientos y declarativos

Los lenguajes de alto nivel pueden ser divididos en lenguajes de procedimiento y en lenguajes declarativos. los primeros son lenguajes como el FORTRAN, el COBOL, el PASCAL, "C", ADA, etc. es decir, que funcionan mediante procedimientos (PROCEDURES) en los cuales el programador debe especificar exactamente todo lo que tiene que hacer el programa ; en cambio los lenguajes declarativos, no operan con procedimientos, si no mediante estructuras de datos y todas las posibilidades con que cuentan éstas para el desarrollo de un programa es mucho más fácil, y el programador se preocupa por centrarse en lo que hay que hacer y no en como hacerlo; Los lenguajes declarativos se hacen cargo internamente de muchos de los detalles de programación, y así el programador, puede escribir un programa solo introduciendo unas cuantas líneas de código y no varias páginas. Un ejemplo de lenguaje declarativo lo constituyen el lenguaje PROLOG, así como todos los lenguajes de cuarta generación, en donde el proceso que establece una tarea no debe escribirse de forma explícita.

VIII.2- Lenguajes de propósito especial y general

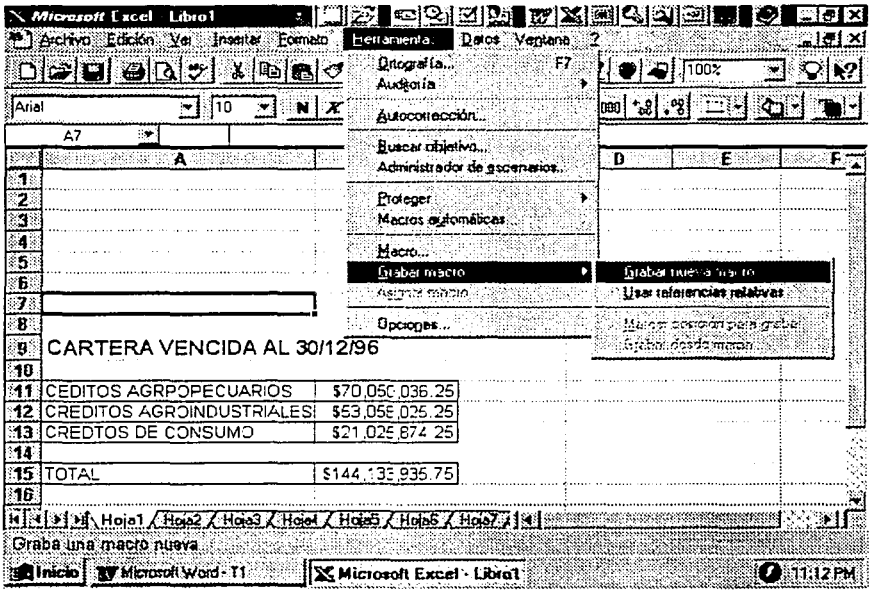
Existe otra forma de dividir a los lenguajes de programación, ésta es mediante la utilización de lenguajes de **propósito especial** y de **propósito general**; como su nombre lo indica, los lenguajes de propósito especial son aquellos enfocados a la resolución de problemas de un mismo tipo, como ejemplo tenemos que para resolver problemas de tipo científico, los programadores utilizan el FORTRAN, mientras que el COBOL es muy utilizado en el ámbito empresarial y para la educación básica y media y los profesores prefieren iniciar casi siempre con BASIC. Los lenguajes de **propósito general** son utilizados para la resolución de todo tipo de aplicaciones, ya sea en el campo científico, educacional, empresarial, o de ingeniería, y por sus grandes cualidades son muy largos y de difícil aprendizaje; entre los mas comunes lenguajes de propósito general se encuentran el ADA, pl/1 y ALGOL 68.

VIII.3-Lenguajes de macros

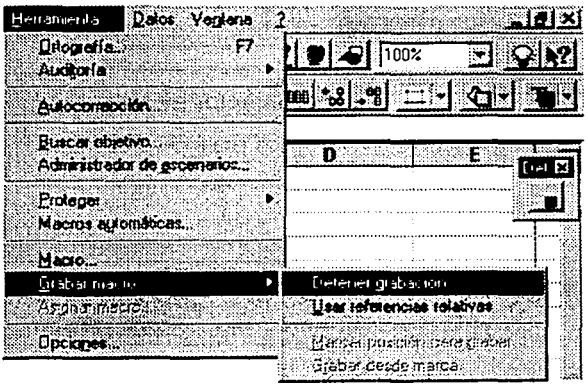
Un lenguaje de macros, o también llamado de guiones es aquel lenguaje que esta diseñado para que usuarios de computadoras inexpertos en la programación, tengan la facilidad de automatizar tareas repetitivas, como por ejemplo dentro de una hoja de calculo (EXCEL, 123 LOTUS, ETC.) o dentro de alguna otra aplicación ; esto se hace desde cualquier paquete que tenga las librerías de "Macros" mediante la activación del inicio de una nueva macro y después solo basta realizar los pasos que comúnmente hacemos dentro de nuestra hoja de cálculo ó aplicación, como: insertar, copiar, crear fórmulas, dar formato, buscar datos, etc. finalmente se procede a terminar la macro y automáticamente todo, lo que hicimos en nuestra aplicación quedará grabado linea a linea en un programa aparte.

El usuario entonces puede examinar la macro y editarla, para rediseñarla a su modo, o correrla directamente sobre su aplicación.

En la siguiente figura se ilustra el método para empezar a construir una macro automática en EXCEL:



Se empieza por poner los datos que deseamos en nuestra hoja de cálculo, y posteriormente se elige "grabar nueva macro", y a partir de este momento se empiezan a realizar los pasos o acciones que comúnmente repetimos en nuestro trabajo, finalmente se elige la opción "detener grabación" y nuestra macro automática quedará grabada.



Finalmente podemos editar y/o correr nuestra macro desde el menú de herramientas.

Un ejemplo del código de una macro que da formato a un archivo desde Excel se muestra a continuación:

Macro realizada en Excel 5.0 con código de Visual Basic Script:

```
Proced Da_Formato_IntS_Bit()
    Ventanas("Ints_Bit.dbf").Activar
    ' Proporciona formato a IntS_Bit.dbf
    ' Elimina 1er. renglon de encabezados
    Filas("1:1").Seleccionar
    Selección.Eliminar Desplazar:=xlHaciaArriba
    'Elimina M y N
    Columnas("M:N").Seleccionar
    Selección.Eliminar Desplazar:=xlHacialzquierda
    ' Selecciona el rango a formatear
    Rango(Celdas(1, 1), Celdas(XTot_D1, 18)).Seleccionar
    Con Selección.Bordes(xlIzquierda)
        .Grosor = xlDelgado
        .IndiceColor = xlAutomático
    Fin Con
    Con Selección.Bordes(xlDerecha)
        .Grosor = xlDelgado
        .IndiceColor = xlAutomático
    Fin Con
    Con Selección.Bordes(xlSuperior)
        .Grosor = xlDelgado
        .IndiceColor = xlAutomático
    Fin Con
    Con Selección.Bordes(xlInferior)
        .Grosor = xlDelgado
        .IndiceColor = xlAutomático
    Fin Con


    ' Centra y elimina los ceros de mas en columna 5
    numfila = 1
    Mientras Celdas(numfila, 5).Valor <> ""
        Si Celdas(numfila, 5).Valor = 0 Entonces
          Celdas(numfila, 5).Valor = ""
        Fin Si
        Celdas(numfila, 5).Seleccionar
        Selección.AlineaciónHorizontal = xlCentrar
        numfila = numfila + 1
    FinMientras

    ' Da formato numérico a columnas 6,7 y 8
    Rango(Celdas(1, 6), Celdas(XTot_D1, 8)).Seleccionar
    Selección.FormatoNúmero = "#,##0.00_);[rojo](#,##0.00)"

    ' Da formato a de porcentaje a columna 9 y 10
    Rango(Celdas(1, 9), Celdas(XTot_D1, 10)).Seleccionar
    Selección.Modelo = "Porcentual"
    Selección.FormatoNúmero = "0.0000%"
    Selección.AlineaciónHorizontal = xlCentrar

Fin Proced
```

VIII.4.- LENGUAJES DE 4TA. GENERACIÓN

Hoy en día mucha gente ve la necesidad de acercarse mas a una computadora, ya sea un estudiante, un administrador, o un empresario; quienes necesitan de alguna forma satisfacer sus necesidades diarias de trabajo, sin embargo este tipo de personas no conoce a fondo la ingeniería de la programación, por lo que los lenguajes de 4ta. Generación vienen a resolver en parte este tipo de problemas debido a que son lenguajes mucho mas accesibles para usuarios que desconocen las estructuras de control y las especificaciones de la programación,

Los lenguajes de cuarta generación son aquellos que se enfocan a resolver el problema de una manera implícita, es decir, se concentran en lo **que hay que hacer**, y no en **como hay que hacerlo**, por lo que los usuarios de este tipo de lenguajes incrementan su productividad de forma considerable. Este lenguaje se encarga de muchos de los detalles que se tendrían que programar en un lenguaje de procedimientos (BASIC, PASCAL, "C", Etc.) ya que pueden obtener resultados con solo teclear unas cuantas líneas y no todo un programa de varias páginas de código.

VIII.4.1.- Características de los lenguajes de cuarta generación :

- Utilizan palabras o frases cada vez mas parecidas al idioma ingles para ejecutar algún comando o instrucción.
- Su facilidad de manejo lleva de la mano a cualquier usuario, por lo que puede ser utilizado no solamente por programadores profesionales, si no por casi cualquier tipo de usuario.
- No operan por procedimientos, como lo hacen los lenguajes de alto nivel, debido a que se enfocan solo a la solución del problema, sin embargo, si un programador desea programar mediante procedimientos, también lo podrá hacer.
- Determina la manera de trabajar de forma automática, es decir, no es necesario especificar todos los detalles del proceso; por ejemplo para crear una ventana de presentación, el formato se puede especificar mediante preguntas que nos hace la máquina, como son el color, el tamaño, los bordes, fuentes, etc.

- Su mantenimiento es más fácil de realizar, debido a que ocupa menos líneas de código y sus estructuras de control son mas entendibles.

VIII.4.2.- Categorías de lenguajes de cuarta generación

Dentro de los lenguajes de cuarta generación existen diferentes categorías, por lo que los podemos clasificar en:

- Lenguajes de consulta y recuperación
- Lenguajes generadores de informes
- Lenguajes generadores de aplicaciones.

VIII.4.2.1- Lenguajes de consulta y recuperación

Los lenguajes de **consulta y recuperación** (Query and Retrieval), son aquellos en donde el usuario puede solicitar y acceder a una base de datos información de una manera muy fácil, mediante preguntas estructuradas. Un lenguaje de consulta sirve como interfaz entre el usuario y la base de datos, y en donde el usuario ya no tiene que escribir largas líneas de procesos y especificar exactamente el formato de salida. Algunos tipos de lenguajes de consulta y recuperación permiten actualizar registros, es decir pueden introducir, cambiar y borrar datos dentro de nuestras bases de datos.

SQL (Structured Query Languaje: Lenguaje Estructurado para Consultas) Es uno de los mejores lenguajes de consulta y recuperación, por lo que los programadores lo han vuelto el mas popular en su tipo. También existe el QBE (Query By Examples: Consulta por Ejemplos) el cual funciona mediante preguntas que plantea el programador mediante el despliegue de la base de datos en forma de una tabla, en donde el programador puede filtrar, comparar y obtener totales de su información.

Los lenguajes consulta y recuperación mas conocidos son:

LENGUAJE	PROVEEDOR
INTELLEC	Artificial Intelligence CORP
On line English	Culliname
Query By Example	IBM
Quick Query	Caci
SQL	IBM

VIII.4.2.2- Lenguajes generadores de reportes

Los lenguajes generadores de reportes o informes funcionan de manera similar que los generadores de consultas, sin embargo estos no pueden acceder a las bases de datos para ser actualizadas, si no únicamente obtienen de ellas los datos necesarios para presentarlos a la salida (Pantalla o Impresora) con un mejor control en cuanto a su formato y presentación. Este lenguaje también obtiene un reporte de toda o una parte del archivo o de la base de datos en cuestión, y el programador puede tomar las características que por default el lenguaje le dará al informe o si lo prefiere puede personalizar la salida del informe mediante pies de página, numeración de las hojas, tabuladores, márgenes, bordes, etc.

Los lenguajes generadores de informes mas conocidos son:

LENGUAJE	PROVEEDOR
Easytrieve	Pansophic
Nomad	NCSS
GIS	IBM
Mark IV	Informics

VIII.4.2.3- Lenguajes generadores de aplicaciones

Los lenguajes generadores de aplicaciones son mas potentes y completos que los lenguajes de consulta y reportes, debido que con este tipo de lenguaje no solo se puede modificar y consultar una base de datos, si no también nos permite desarrollar todo un sistema en donde podamos realizar cálculos, obtener y guardar información en archivos, validar datos, realizar altas, bajas, consultas, etc. Todo esto se realiza en un muy alto nivel de programación, es decir, en donde no es necesario especificar los procedimientos. Algunos lenguajes generadores de aplicaciones producen un código fuente, en donde el programador si lo desea puede editar y modificar dicho código para mejorar y personalizar más su aplicación, otros solo producen parte de ese código, por lo que el programador lo puede enlazar con otras aplicaciones producidas por él o por el mismo generador. Hemos visto como este tipo de lenguaje produce resultados mas variados que los demás, sin embargo esto implica que el usuario deba tener un conocimiento mas amplio en la construcción de software.

Lenguajes generadores de aplicaciones:

LENGUAJE	PROVEEDOR
ADS	Cullinet
Application Factory	Cortex Corporation
FOCUS	Information Builders
MAPPER	Sperry
MANTIS	CINCON
NATURAL	Software AG
RAMIS	Mathematica Inc

IX.- LENGUAJES E IMÁGENES

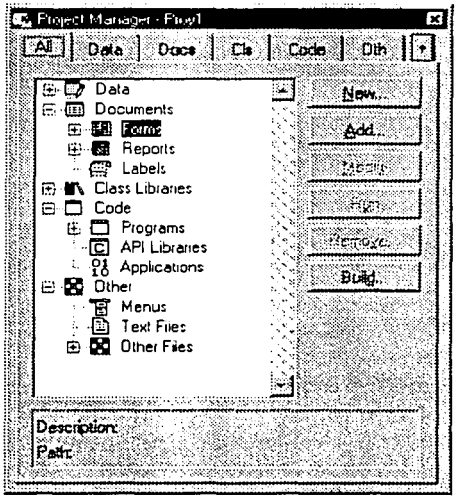
Actualmente existen lenguajes en donde no se escribe bajo la tradición de tener un editor de textos y empezar a ingresar nuestro código, si no más bien nos ofrecen la posibilidad de trabajar con herramientas de programación visual, que nos permitan crear programas en un ambiente de imágenes, ventanas y menús, es decir, los programas finales tendrán dibujos y objetos que podremos observar en la pantalla y mediante un click se puede llamar a otra ventana o aplicación.

Estos lenguajes proporcionan fantásticas herramientas para construir documentos interactivos, de capacitación, entretenimiento, etc. y muchos de ellos están diseñados para que los programadores novatos puedan crear software interactivo, sin tener que dominar al 100% las estructuras de control y las reglas de la compleja programación.

Dentro de este tipo de lenguajes tenemos el VISUAL BASIC y VISUAL FOX los cuales están siendo utilizados con mucha popularidad en el mercado, pues mediante sus herramientas de programación visual hacen posible que los programadores logren hacer trabajos profesionales con un menor tiempo y esfuerzo.

Los lenguajes visuales están centrados en dos tipos de objetos, ventanas y controles, que permiten diseñar sin programar, una interfaz gráfica para una aplicación. Para realizar una aplicación se crean ventanas, llamadas formularios, y sobre ellas se dibujan otros objetos llamados controles, tales como cajas de texto, botones de ordenes, listas desplegables, etc. Cada objeto (ventanas y controles) está ligado a un código que permanece inactivo hasta que se de el suceso que lo activa. Por ejemplo, podemos programar un botón que de ordenes (objeto que se puede pulsar) para que responda a un clic del ratón.

En la siguiente figura tenemos el administrador de proyectos de Visual Fox, el cual contiene los formularios, reportes, programas, menús y tablas utilizadas en el desarrollo de nuestro sistema.



Existen otras herramientas para la programación visual como lo son: Toolbook, NextStep, LinkWay, HiperStudio, VisualStudio etc. que también son muy poderosas y mediante las cuales podemos manejar menús, abrir cajas de diálogo, dar mantenimiento a información, mandar a impresión nuestra información, etc.

X.- PROGRAMACIÓN Y SISTEMAS EXPERTOS

Hablar de sistemas expertos significa estar en el área de la inteligencia artificial, por lo que empezaremos por describir lo que representa ésta rama:

La Inteligencia Artificial ha tenido varias definiciones en su corta vida (empezando con Alan Turing en 1940), pero la que encaja más en la actualidad es:
La Inteligencia Artificial es el estudio de las computadoras, para poder hacerlas percibir, razonar y actuar. Esto implica lo mas difícil con lo que hasta ahora el hombre tenga como reto en el área de la investigación, debido a que la máquina mas sofisticada, hecha hasta este momento, no tiene comparación alguna con el procesamiento en paralelo que puede realizar el cerebro humano.

Tal vez las máquinas superen al hombre en tareas repetitivas o peligrosas, pero en cuanto a la toma de decisiones o de intelecto, no se ha podido lograr este objetivo plenamente. Para poder lograr estos objetivos, los investigadores han seguido dos caminos:

1.- han tratado de investigar el comportamiento del cerebro humano, para poder aplicarlo de manera similar a los circuitos electrónicos de las máquinas.

2.- Han tratado de llevar una investigación independiente de lo que es el cerebro humano, para hacer que una máquina tome decisiones congruentes.

Problemas que implica el primer camino:

- Intenta simular los procesos mentales humanos, con las máquinas, sin embargo este proceso resulta engorroso, debido a que no se pueden simular las múltiples opciones humanas y pensamientos inconscientes, ideas o intuiciones instantáneas con que el hombre pudiera solucionar un problema, y de ahí que el investigador lo pudiera traspasar a un modelo de software.

- Como se dijo anteriormente, existe gran diferencia entre lo que es la estructura del cerebro humano, y la de una computadora, por lo que la computadora más potente no podría acercarse al funcionamiento en paralelo del cerebro humano, es decir, dividir un proceso en tareas mas pequeñas para poder realizarlas mas fácilmente y al mismo tiempo

El segundo camino afirma que la inteligencia humana es solo uno de varios tipos de inteligencia, por lo que trata de resolver los problemas de forma distinta a los del cerebro humano. Así por ejemplo, los primeros experimentos que hizo el hombre para poder volar, trataba de imitar a las aves y lo único que consiguió fueron severos accidentes.

Sistema basado en conocimientos o Sistema experto.

Un Sistema Experto es aquel que se basa en un cumulo de hechos llamados bases de conocimiento, para un proceso de toma de decisiones.

Las bases de conocimiento son una colección de conocimientos especializados en una determinada rama de la industria, medicina, navegación, etc. son construidas por los ingenieros de software mediante entrevistas con gente experta en determinada rama, considerando como experto a aquella persona que posee gran cantidad de conocimiento en determinada área.

Algunos de los últimos sistemas expertos pueden incrementar y cambiar sus bases de conocimientos para poder mantenerse actualizados, mediante la observación de la toma de decisiones de los expertos humanos. Para la obtención del conocimiento el ingeniero de software que diseña el programa, no solo obtiene información del experto humano, si no también de bases de datos estadísticas, políticas de las empresas, reglamentos gubernamentales, índices de la bolsa de valores, etc.

Los sistemas expertos basan en conseguir sus resultados mediante la navegación en su información a través de instrucciones SI - ENTONCES

XI.- EL FUTURO DE LA PROGRAMACIÓN

Nadie sabe con exactitud acerca de cómo serán los lenguajes del futuro, o cómo se programará una aplicación futurista. Por lo pronto estamos consientes de que la Programación Visual, los Lenguajes Orientados a Objetos y los Lenguajes de Cuarta Generación están tomando mucha fuerza en todas las áreas de aplicación. Sin embargo es probable que el futuro de la programación siga las siguientes tendencias:

1.- Mucho se piensa que la programación orientada a objetos POO será el futuro de la programación, debido a que con ella es posible construir programas a partir de objetos con propiedades que nos permiten enviar mensajes entre si.

2.- Los siguientes lenguajes de programación se acercarán más a los lenguajes naturales, como el ingles. Debido a que aún hoy el lenguaje mas avanzado se encuentra limitado a ciertas reglas y le falta inteligencia propia, por lo que la inteligencia artificial tendrá un papel importante en los próximos años.

3.- Es probable que casi cualquier persona pueda desarrollar un programa de aplicación, y no será necesario que conozca la complejidad de un lenguaje de programación técnico, es decir, se dejará el poder de la programación en manos de los usuarios.

4.- Las herramientas de la programación visual tendrán un papel importante en el desarrollo de los sistemas, ya que mediante iconos, herramientas de dibujo, menús, ventanas de diálogo, etc. proporcionan elementos prácticos que permiten programar mas fácilmente.

5.- Es posible que las herramientas de programación del futuro realicen aplicaciones completas a partir de una descripción completa del problema, y que ellas mismas proporcionen el mantenimiento adecuado a las nuevas necesidades e incluso anticiparse a nuevas contingencias sin ninguna intervención del programador.

6.- Los desarrollos WEB por internet ya están revolucionando el funcionamiento del mundo electrónico, por lo que se espera que continúe esta tendencia.

Lo que es seguro decir es que los lenguajes de programación del futuro serán muy distintos a los lenguajes de alto nivel actuales , y que el personal que trabaje con ellos comparará e los antiguos lenguajes como nosotros lo hacemos con las tarjetas perforadas.

Resumen

Afortunadamente para todos los desarrolladores, los lenguajes de programación han evolucionado a través de varias generaciones, cada una de ellos más fácil de usar y más poderosos que sus predecesores. Ya casi no se programa en lenguaje de máquina, pues es primitivo y difícil; en cambio los lenguajes de alto nivel, se parecen mas al inglés, por lo que su uso es mas simple que el lenguaje de máquina o el lenguaje ensamblador; también pueden presentar grandes beneficios en la portabilidad y mantenimiento de los mismos, pues trabajan independientemente al tipo de procesador que utilice la máquina.

Casi cualquier lenguaje de programación moderno ayuda a programar en forma estructurada, pues contienen las estructuras básicas de control: secuencia, selección y repetición, las cuales ayudan a producir menos errores y a llevar un mejor control y entendimiento del programa. Sin embargo, si lo que se busca es hacer mas eficiente nuestra aplicación, entonces se puede recurrir al lenguaje C, el cual nos permite interactuar ditrectamente con los recursos de la máquina.

También existen aplicaciones con lenguajes de macros y lenguajes de consultas que ponen el poder de la programación en manos de programadores inexpertos. Los lenguajes de consultas son lenguajes de cuarta generación que no operan por procedimientos; es decir, permiten que los programadores se concentren en la definición de la tarea y no en los pasos que implica la resolución de la misma. Utilizando las herramientas de la programación visual el programador puede usar iconos, herramientas de dibujo, menús y ventanas de código para construir programas sin tener que escribir demasiado código. Las herramientas de programación orientada a objetos (POO) hacen posible construir programas a partir de objetos con propiedades y la capacidad para enviar mensajes entre si, por lo que se piensa que la programación orientada a objetos representa la panacea en el futuro de la programación.

Capítulo
DOS

CALIDAD DE LOS DESARROLLOS

Cuando se construye un edificio o un puente, los ingenieros diseñan sus estructuras en base a principios y técnicas existentes que tratan de asegurar que las construcciones no se vendrán abajo. La ingeniería de software es una rama de las ciencias de la computación que trata de aplicar principios y técnicas de la ingeniería, para la construcción de software, sin embargo esta rama es relativamente nueva y no existe un método infalible para asegurar que nuestro software sea 100% confiable.

La ingeniería de software esta íntimamente ligada a las disciplinas convencionales de la ingeniería : así como en la construcción de un automóvil deportivo que requiere todas las restricciones de aerodinámica, velocidad y estabilidad, la ingeniería de software se apoya en métodos, herramientas y procedimientos para el buen desarrollo de un sistema y que éste cumpla con ciertas características de costo, duración del proyecto, tiempo de respuesta, portabilidad, y "confiabilidad".

XII.- CONCEPTOS DE INGENIERÍA DE SOFTWARE

La Ingeniería de Software se define como la disciplina tecnológica preocupada de la producción sistemática y mantenimiento de los productos de software que son desarrollados y modificados en tiempo y dentro de un presupuesto definido. La Ingeniería de Software difiere de la programación tradicional en que se utilizan técnicas de ingeniería para especificar, diseñar, instrumentar, validar y mantener los productos dentro del tiempo y el presupuesto establecidos para el proyecto ; además esta ingeniería se preocupa por aspectos administrativos que quedan fuera del dominio normal de la programación. Los Ingenieros de Software a diferencia de los programadores, se ocupan de aspectos como el análisis, el diseño, la verificación y prueba de programas, la documentación, el mantenimiento y la administración del proyecto ; todo esto enfocado a producir sistemas de calidad.

Dentro de la Ingeniería de Software tiene un peso predominante la "Calidad de los Desarrollos", y ésta se verá reflejada hasta el término de cualquier desarrollo; dentro de la palabra calidad deben estar implícitos los siguientes conceptos y en el siguiente orden:

a).- Confiabilidad,
b).- Utilidad
c).- Portabilidad
d).- Eficiencia
e).- Claridad

XII.1.- Confiabilidad

El factor más importante en la calidad de un producto es su grado de confiabilidad, debido a que independientemente de los demás factores, este punto es medular en cualquier sistema pequeño o un gran sistema militar o médico. La confiabilidad de un producto está definida por la capacidad de una aplicación para desempeñar una función requerida bajo ciertas condiciones durante un tiempo específico. Por lo que de nada sirve que un sistema sea utilizado, si éste no desempeña adecuadamente la función para el cual fue diseñado o no arroja los datos correctos, lo cual provocará irritación por parte de los usuarios, reprocesos, pérdidas millonarias, o hasta la pérdida irreparable de vidas humanas. La cantidad de esfuerzo que se debe asignar a la confiabilidad debe ser proporcional

al costo derivado de las imperfecciones que pudiera ocasionar el sistema, por lo que debe existir un nivel mínimo de confiabilidad en el desarrollo de cada producto.

XII.2.- Utilidad
Teniendo confiabilidad, se puede decir que un programa será útil, es decir, una vez que se halla verificado que el programa satisface las necesidades del usuario y que realiza las funciones esperadas, entonces el producto tomará las características de "Utilidad". Hasta este punto la planeación cuidadosa, el análisis detallado y la participación del cliente son básicas para la obtención de productos confiables y útiles.

XII.3.- Portabilidad
En algunas ocasiones la Portabilidad de un producto entre las diferentes máquinas de la empresa es de importancia capital, o que nuestro sistema corra bajo diferentes ambientes de software representa un aspecto relevante para los clientes, pues hay ocasiones en que una compañía cuenta con diferentes equipos de cómputo y es imprescindible que todas las terminales cuenten con un determinado sistema.

XII.4.- Eficiencia
Un producto de programación debe ser **eficiente** respecto a las características particulares del equipo, siempre y cuando la aplicación lo amerite, en los inicios de la informática, el Hardware era muy limitado, por lo que había que optimizar y eficientar todos los recursos de la memoria. Afortunadamente en la actualidad, los grandes avances de la electrónica nos han permitido dar una mayor prioridad a otras características de calidad, no obstante existen programas que ameritan un dedicado esfuerzo en la parte de optimización de recursos (Sistemas en tiempo real corriendo en una microcomputadora ó sistemas medianos de grandes bases de datos) que están críticamente limitados por el tamaño de la memoria o que los accesos y la rapidez a los datos, requieren una velocidad de ejecución mayor, por lo que para estos casos la eficiencia será una de los principales atributos de nuestra aplicación.

XII.5.- CLaridad
Un punto que el usuario no puede observar tan fácilmente es la **claridad** con la que se escribe el código que genera la aplicación, pero que ayuda da manera importante al programador a realizar cualquier mantenimiento preventivo o correctivo ; debido a lo costoso que es el mantenimiento en tiempo, dinero y esfuerzo, la clave de éste radica en hacerlo comprensible, es decir, escribirlo con claridad y hacerlo fácil de entender. Los productos de programación que se entregan a los usuarios deben tener una integridad conceptual y ser claros en el propósito del proyecto.

XIII.- IMPORTANCIA DEL SOFTWARE

Los ingenieros de software han logrado avances importantes en el desarrollo de sistemas, pues hoy en día programan mucho más fácil que hace una década. El avance del hardware ha sido mucho mayor y el principal reto es la calidad del software, las actuales técnicas de desarrollo no aseveran que un sistema de software sea infalible durante todas las circunstancias y mientras más dependamos del software en nuestra vida diaria más importante será que los ingenieros de la computación encuentren formas de como crear software en el cual podamos confiar. Todas las organizaciones ya dependen de un sistema de información computarizado, sin el cual sería imposible de trabajar si éste faltara, debido al gran volumen de información que manejan empresas del tipo empresarial, militar, turístico, industrial, etc. Estas presiones de manejar gran volumen de información y la rapidez con la que se requieren procesar los datos han provocado una reorganización en los métodos empleados para la construcción de software, para satisfacer estándares muy estrictos en el control de la calidad, de acuerdo a una planificación, control y presupuesto adecuado.

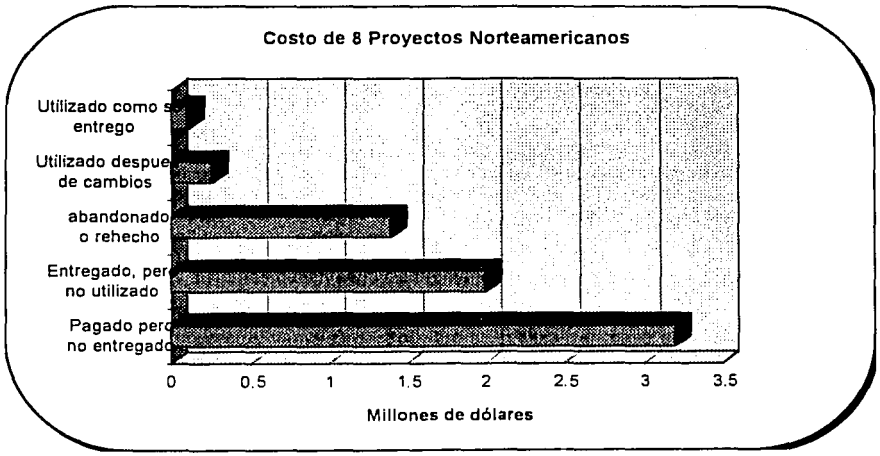
XIII.1.- Perfil del personal de sistemas

Las personas que desarrollan software deben trabajar en equipo y es deseable que una lógica formal de programación, que tengan gran capacidad de razonamiento y entender a detalle el funcionamiento de lo que se va a automatizar, así como poseer conocimientos de administración, contabilidad, matemáticas y economía ; cada equipo debe tener a una persona comprometida a verificar que el trabajo realizado por el equipo sea el correcto, esté estructurado con propiedad, disponga de las interfaces apropiadas con el software de otros equipos y se desarrolle de acuerdo a lo previsto en tiempo y costo. Como podemos ver la ingenieria de software requiere de los ingenieros un amplio rango de conocimientos y de gestión, ya que abarca el ciclo completo del desarrollo de un sistema, que va desde la determinación del problema, hasta el mantenimiento y el trabajo se debe hacer buscando un alto grado de estandarización, un bajo costo durante el ciclo de vida y una máxima confiabilidad.

XIII.2.- El costo del Software .VS. Hardware

a través del tiempo el costo del Hardware ha ido disminuyendo y gracias a la microelectrónica existen máquinas cada vez mas potentes, rápidas y de bajo costo, sin embargo no podemos decir lo mismo del software , el cual ha tenido muchos problemas en el costo de su desarrollo, pues muchos sistemas tardan más tiempo de lo previsto y resultan sumamente costosos ; muchas veces el software que se desarrolla es abandonado o simplemente no se utiliza después de terminado tiempo, por lo que el resultado del software se aleja de la línea ideal y los ingenieros que desarrollan grandes sistemas deben seguir enfrentando el problema de confiabilidad y altos costos de desarrollo

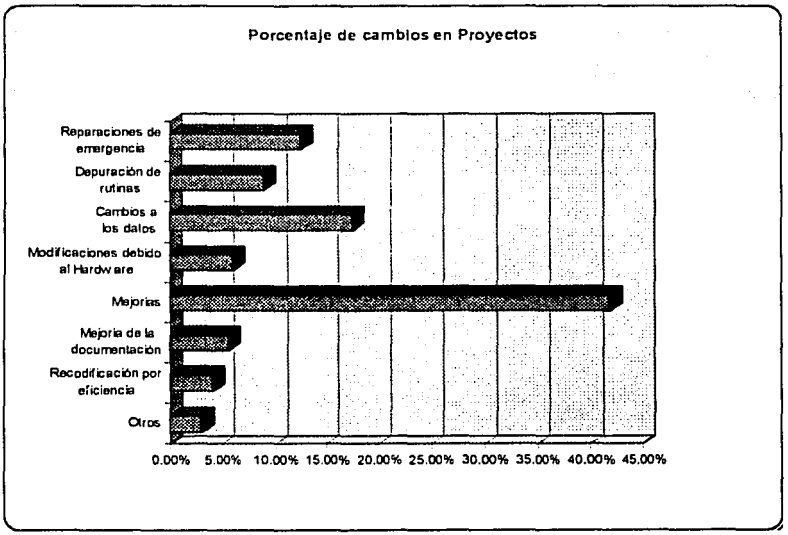
Todos los proyectos formales de Software se realizan en base a un estudio de tiempos y costos, sin embargo casi muy pocos llegan a liberarse mediante condiciones óptimas, es decir, estos proyectos sufren descalabros durante su desarrollo o bien durante su término; un indicador de esto lo podemos ver en la siguiente gráfica, que representa el costo de 8 macroproyectos Norteamericanos de defensa. No olvidemos que estas cifras representan proyectos muy grandes, y que para proyectos medianos en cualquier empresa, su valor resulta mas bajo, pero aún así, estos costos son significativos.



El primer factor que interviene en el costo elevado de los proyectos es que el Software es intangible, por lo que no lo podemos ver, si no hasta que ejecutamos la aplicación ya terminada, por lo que en este punto el proveedor y el cliente se dan cuenta que las expectativas esperadas son muy diferentes a la realidad, por lo que se procede a realizar mejoras o mantenimientos con un alto costo de desarrollo.

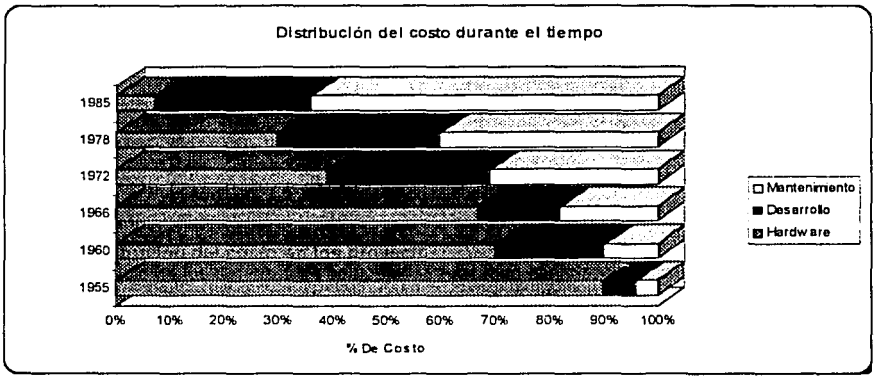
Otra razón que interviene en la inflación del Software es la complejidad con que es desarrollado , es decir, cuando existe alguna modificación al programa, este puede resultar sumamente tedioso al tratar de comprenderlo y aún mas modificar cualquier parte de él, por lo que muchos programadores prácticamente le dan la espalda y finalmente puede ser abandonado.

Los sistemas de Software ideales son aquellos que satisfacen plenamente las necesidades del cliente, dentro de un mundo real, equilibrando la funcionalidad con la eficiencia del mismo programa, por lo que otro factor determinante en el costo del Software son los cambios a los cuales se tienen que adaptar los sistemas debido a cambios en las leyes fiscales, a las fusiones de la compañía con otras empresas, debido a la inserción de nuevos productos en el mercado, debido a mejoras, etc. en fin un sistema esta sujeto a sufrir un sin número de cambios continuos desde su apertura, e incluso desde su desarrollo. En el mejor de los casos un tratamiento adecuado de estos cambios puede llegar a representar hasta un 75% del total de presupuesto para el proyecto, y en otras ocasiones el proyecto puede llegar a ser cancelado debido a los gastos excesivos de mantenimiento.



XIII.3.- Distribución del costo en los proyectos

Durante el inicio de los sistemas de computo, se gastaban cantidades considerables en la utilización del Hardware, sin embargo en la actualidad, gracias a los circuitos integrados y al alto nivel tecnológico que se han desarrollado en las máquinas , se ha reducido drásticamente el costo del Hardware y ya en estos años 90's solo ocupa menos del 10% del presupuesto total. Por otra parte, los costos de los desarrollos se han venido incrementando debido a la complejidad del los sistemas y por ende, esto exige la inversión de gran cantidad horas- hombre por lo que los costos del personal de desarrollo se han incrementado y las labores de mantenimiento aumentan debido a que se van acumulando en el transcurso del tiempo más sistemas de programación.



XIII.4.- Factores principales en la calidad, productividad y en el costo del Software

XIII.4.1.- Capacidad del programador

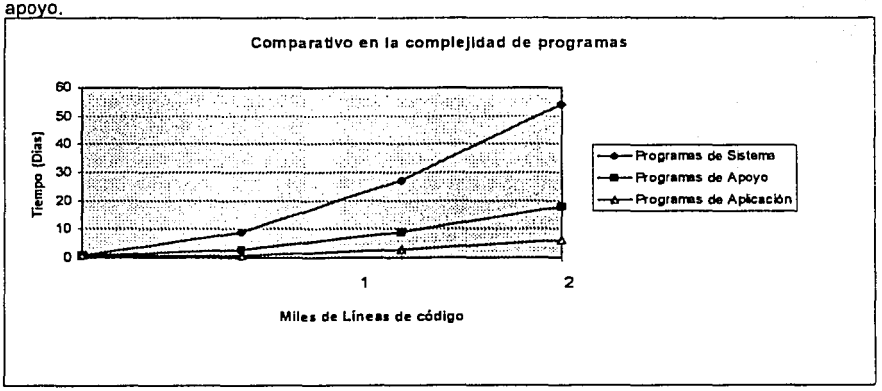
La calidad y la productividad están en función directamente de la capacidad individual de las personas, y ésta se puede dividir en dos aspectos : Capacidad de competencia global de cada individuo y capacidad en un entorno familiar de aplicación, la primera se refiere a la habilidad de desarrollar proyectos (pantallas, menús, informes, programas) en un corto tiempo y con una alta calidad debido a un alto grado de inteligencia del individuo que desarrolla las aplicaciones ; por otro lado, el desenvolverse dentro de un marco conocido para el programador, resulta mas rentable para el proyecto, ya que los tiempos de capacitación se reducen y se programa bajo un marco mas seguro, es decir, si un programador que esta acostumbrado a trabajar con empresas financieras y bases de datos cambia su entorno y desarrolla aplicaciones científicas, entonces su desempeño se reducirá drásticamente. En proyectos muy grandes puede existir un equilibrio , debido a que las diferencias individuales pueden limar esas diferencias en la entrega oportuna de trabajos, sin embargo en proyectos pequeños, la capacidad individual es un factor determinante. La ingeniería de software ha tratado de contrarrestar estas diferencias proporcionando herramientas y técnicas que permitan desarrollar trabajos profesionales y competentes.

XIII.4.2.- Comunicación en el equipo de desarrollo

Anteriormente se consideraba que la programación era una actividad personalizada y que se requería contacto social solo para lo necesario, sin embargo se debe tener una estrecha y vinculada relación con todo el proceso completo, para entender de forma general el funcionamiento de todo el proyecto. Algunos programadores en raras ocasiones analizan los detalles exactos de su trabajo de manera sistemática, por lo que pueden existir malos entendidos en sus procesos o subrutinas y así cometer errores que puedan ser detectados hasta tiempo después. Otro factor que es causa de la mala comunicación y por ende ocasiona improductividad es un gran tamaño del producto, debido a que existen más programadores y más módulos que tienen que interactuar entre sí y su complejidad también va en aumento. Todo lo anterior aplica para proyectos dependientes, pero para proyectos con tareas independientes, el programador se puede dedicar únicamente a recibir los datos de entrada, procesarlos y dispararlos, para que el siguiente módulo independiente los ocupe.

XIII.4.3.- Complejidad del producto

Es imprescindible que para buscar **productividad** en cuanto a la obtención de programas a corto plazo, debamos tomar en cuenta lo que se va a desarrollar ; si lo que queremos es tener un programa de aplicación de altas y bajas de registros, es evidente que resultará menos tedioso y mucho más productivo (En cuanto al número de líneas escritas de código) que desarrollar programas de sistema operativo que interactúen con los recursos del sistema. Los programas de apoyo son tres veces más difíciles que los programas de aplicación, pero los programas de sistema suelen ser tres veces más complejos que los de apoyo.



XIII.4.4- Notaciones apropiadas

La representación gráfica de cualquier proyecto ayuda a identificar y entender las diferentes partes que componen al proyecto, sin embargo, a diferencia de la Ingeniería electrónica o civil, no existe en la Ingeniería de Software una notación fija universal a seguir, por lo que este beneficio se verá reflejado solamente cuando en todas las organizaciones e industrias se adopte un conjunto de notaciones bien definidas para los proyectos.

XIII.4.5- Control de cambios

En un mundo en donde la única constante es el cambio, los líderes de proyectos deben establecer procedimientos en donde se formalice la petición de cualquier modificación al programa y poder determinar la mejor forma de realizar el cambio y que este quede debidamente documentado.

XIII.4.6- Nivel tecnológico

La estabilidad y disponibilidad del ambiente computacional (Hardware y Software) influyen directamente en la productividad y en la calidad de los desarrollos, pues el contar con buen software de programación nos proporciona características mejoradas para la definición y manejo de datos, estructuras de construcción para la definición del flujo de control, mejores facilidades de modularización, manejo eficiente de condiciones, facilidades para la programación concurrente, así como herramientas para la administración y control del desarrollo del proyecto; Algunos lenguajes modernos brindan características adicionales para mejorar la productividad y la calidad del producto, entre estas características están la verificación de tipos de datos la abstracción de datos, la compilación separada, el manejo de excepciones y de interrupciones, todo lo anterior nos proporciona mejor calidad en los desarrollos, mientras que la velocidad y tiempos de respuesta del Hardware nos proporcionan una mayor productividad.

XIII.4.7.-Nivel de confiabilidad

Así como en alguna fábrica se presentan productos que cumplen con niveles básicos de calidad, entonces todo producto de programación debe poseer un nivel elemental de confiabilidad, y éste debe empezar y continuar desde el análisis, diseño, desarrollo, pruebas y mantenimiento del producto programado.

La confiabilidad de un producto se define como la probabilidad de que un programa funcione de acuerdo a las especificaciones convenidas durante cierto tiempo y el nivel de confiabilidad se mide en base a los impactos que el programa pueda ocasionar por alguna falla :

Niveles de confiabilidad:

Categoría	Consecuencia de falla
Muy bajo	Alguna molestia menor
Bajo	Pérdidas fáciles de recuperar
Normal	Dificultad relativa en la recuperación
Alto	Gran pérdida económica
Muy alto	Poner en riesgo una vida

Con todo lo anterior se debe asegurar una mayor confiabilidad y calidad en los desarrollos, sin embargo se pierde un poco de productividad medida en solo en términos de líneas de código producido en un mes, lo cual es un costo que definitivamente se tiene que asumir.

XIII.4.8.- Captación del problema

Los problemas principales en los desarrollos son el mal entendimiento del problema y la no inclusión de todas las variables del proceso en la solicitud de desarrollo, pues suele suceder que la persona que realiza el requerimiento o que indica el como automatizar el proceso, no es la misma que lo va a utilizar como usuario final, por lo que se corre el riesgo de que no se especifique el problema correctamente o peor aún, que la naturaleza del proyecto sea de difícil solución y comprensión de manera que la falta de entendimiento del cliente no pueda ser suficiente para obtener la mejor solución. Generalmente los clientes no conocen las capacidades y limitaciones de los equipos de cómputo, ni mucho menos piensan en términos lógicos ni algoritmicos y por otro lado los ingenieros de Programación desconocen y no entienden al área en donde desarrollarán alguna aplicación, por lo que muchas veces tienen dificultad de comunicarse con el cliente, debido a sus antecedentes educacionales, sus distintos puntos de vista y a su experiencia propia. Otro caso no mejor, es cuando la

verdadera naturaleza del problema se detecta hasta que el desarrollo se ha instalado o se encuentra operando.
Por otro lado, una planeación detallada, entrevistas claras con los clientes, problemas bien especificados y documentados y una inspección de la tarea manual por parte del ingeniero de Software pueden incrementar el entendimiento con el cliente sobre el problema a tratar.

XIII.4.9.- Tiempo y recursos disponibles
Puede parecer lógico que un proyecto de programación tarde seis meses con un programador y que el mismo proyecto tarde 1 mes con seis programadores, sin embargo esto no aplica en la ámbito del Software, pues el utilizar a seis programadores retrasaría la fecha de termino debido a la capacitación, a la comunicación necesaria de todos integrantes y a la curva de aprendizaje invertida para cada uno de los seis elementos, por otro lado y dependiendo de la naturaleza del proyecto, el utilizar dos programadores durante tres meses resultaría mas eficiente que uno, debido a la retroalimentación que cada uno reciba del otro. Por lo tanto para determinar el nivel óptimo de personal y el tiempo requerido para desarrollar las diferentes actividades en cualquier desarrollo es un aspecto difícil e importante en la estimación global de costos y recursos, pues como se dice: ***no por poner a trabajar a nueve mamas el niño va a nacer en un mes.***

XIII.4.10.- Habilidades requeridas
El Ejercicio de la ingeniería de la programación requiere de una gran variedad de habilidades, especialidades y cierta experiencia, pues la obtención de la información que proporciona algún cliente, con el propósito de determinar sus verdaderas necesidades requiere de cierta habilidad de entendimiento, de comunicación, de tacto y diplomacia, así como cierta experiencia y conocimiento del área de aplicación. La detección y definición de las necesidades y las actividades de diseño son de tipo conceptual y requieren de una gran dosis de habilidad en la resolución de problemas. También es muy recomendable ser detallista a la hora de implementar el programa, se requiere conocer las reglas de sintaxis y escribirlas en forma perfecta a la hora de compilar , se requieren características detectivescas a la hora de depurar los programas y a la hora de realizar las pruebas se necesita considerar todas las situaciones posibles que se puedan presentar o al menos la mayoría de ellas. El trabajo con administradores, directivos y otros ingenieros exige la capacidad de comunicación oral e interpersonal, por lo que todas estas habilidades pueden ser desarrolladas en un marco técnico y gerencial

XIII.4.11.- Facilidades y recursos
Los programadores mejor motivados, son aquellos que cuentan con una máquina que sea rápida y tenga los últimos adelantos tecnológicos, así como aquellos a quienes reciben capacitaciones constantes y están respaldados por un buen equipo de trabajo; independientemente de el sueldo, la mejor recompensa para un programador esta en la naturaleza misma de su trabajo, es decir el éxito y la frustración se encuentran en sus propios desarrollos. Los buenos programadores sienten que los aspectos positivos de su trabajo son las tareas que representan un verdadero reto, así como las oportunidades de desarrollo profesional, mientras que los aspectos negativos a los que se enfrentan son la ineptitud administrativa, políticas de la compañía mal enfocadas y la burocracia interna.

XIII.4.12.- Formación inadecuada
La mayoría de los egresados de las carreras de computación tienen un entendimiento de las teorías y conceptos básicos de la información y su procesamiento, comprendiendo estos términos en el sentido más amplio posible, mientras que la ingeniería en productos de la programación se encarga del análisis, diseño, programación, pruebas, verificación, documentación, operación y mantenimiento del software y éstas actividades son demandadas ampliamente por las empresas. Los programadores novatos deben de

enfrentarse ante estos retos y desarrollar capacidades para poder enfrentarse con mayor seguridad a las organizaciones que reclaman la utilización de análisis estadísticos e impactos económicos, trabajar con técnicas de administración de proyectos y desarrollar la habilidad del trabajo en grupo.

XIII.4.13.- Habilidades gerenciales

En algunas empresas los proyectos de programación son supervisados por individuos que tienen poco conocimiento informático y pueden mal encaminar los resultados que se desean, por otra parte la promoción a puestos de administración de proyectos a individuos técnicamente competentes, con poca inclinación gerencial y sin entrenamiento administrativo, puede producir también resultados negativos.

Las actividades técnicas y gerenciales son ambas importantes para el éxito y fracaso de una proyecto. Un gerente de proyectos de desarrollo debe controlar los recursos y el ambiente en que las actividades técnicas se suceden, por lo que tienen la responsabilidad de que los proyectos se entreguen a tiempo, dentro del presupuesto estimado y con las especificaciones convenidas, además de encargarse de la elaboración del plan de trabajo, contratación de nuevos proyectos, desarrollo de estrategias de mercado, así como la contratación y entrenamiento del personal; también debe aplicar las actividades de administración de un proyecto como son la estimación de costos, las políticas de asignación de recursos, el control del presupuesto, la definición de logros del proyecto, la medición del avance del proyecto, la reasignación de recursos, los ajustes al calendario de trabajo, el establecimiento de procedimientos para control de calidad, el mantenimiento de las diversas versiones, la documentación de los desarrollos, la promoción de la comunicación entre los miembros del equipo y la comunicación y el desarrollo de acuerdos contractuales con los clientes. Se dice que un Ingeniero en Computación trabaja aislado del grupo administrativo, sin embargo éste puede ser el eslabón de reingeniería muy importante en cualquier área, pues conociendo al departamento y los alcances y limitaciones de los sistemas podría llegar a optimizar todos los procesos existentes, es así que el caso ideal es aquel ingeniero en Ciencias de la Computación, el que se encargue de los proyectos conociendo el rol y responsabilidad gerencial.

XIII.4.14.- Otros factores

Existen otros factores sociales e interpersonales que influyen en los programadores y que repercuten de manera directa en la productividad y la calidad de los desarrollos, así como también el acceso, la estabilidad, el tamaño y la familiaridad que se tenga con las bases de datos y los sistemas de cómputo utilizados y/o la experiencia en el lenguaje de programación utilizado.

XIV.- ASEGURAMIENTO DE LA CALIDAD

La ingeniería de Software está enfocada directamente a generar productos de calidad, y no solo durante las etapas de codificación y pruebas, sino desde el inicio del ciclo de vida de cada sistema. Un producto de alta calidad satisface las necesidades del usuario, se apega a las especificaciones de los requisitos y presenta una ausencia de errores. Para conseguir calidad podemos seguir ciertas reglas que nos ayudarán a tener un mejor aseguramiento de calidad de software.

1.- Los requisitos de nuestros clientes deben ser la base de las medidas de la calidad
2.- Se deben seguir estándares o reglas durante el desarrollo
3.- Se deben tomar en cuenta los requisitos intrínsecos de todo software profesional.

Garantía de la Calidad en el Software (SQA: Software Quality Assurance)

Actualmente el aseguramiento de la calidad en toda empresa, es una actividad de protección que debe ser considerada como parte integral de todo desarrollo, pues en los años 50's, quien garantizaba la calidad de los proyectos se resumía solo al programador, quien era el encargado de realizar el desarrollo, las pruebas y la liberación de su aplicación, esto ha evolucionado conforme los programas se han vuelto mucho más complejos, pues ya existen verdaderos ejércitos en el aseguramiento de la calidad en las grandes empresas (Gestores, programadores, líderes, usuarios, etc.).

La SQA es un modelo planeado y sistemático de todas las acciones necesarias para asegurar la calidad del software y se compone principalmente de las siguientes actividades:
a).- Aplicación de métodos y herramientas técnicas para el análisis, diseño, codificación y pruebas
B).- Realización de revisiones técnicas formales que se aplican durante cada paso de la ingeniería de software
C).- Aplicación de pruebas multiescala
D).- Implantación de procedimientos que aseguren un ajuste a los estándares de desarrollo
E).- Incluir un control en la documentación del software y de los cambios realizados
F).- Implantar mecanismos de medida e información de la aplicación.

XIV.1.- Solicitud de desarrollo:

Es el primer documento que conecta al cliente con el grupo de desarrollo, pues es aquí en donde el cliente plasma su necesidad, la cual puede ser la corrección de un sistema productivo ó el nacimiento de un nuevo proyecto a través de una idea de mejora (Ver anexo 2_1).

XIV.2.- Carta de especificaciones

El software de calidad debe continuar con una especificación rigurosa (ver anexo 2.2) y un diseño bien planeado; una vez que ya tenemos la especificación y prototipo se continúa la revisión de calidad con la ***revisión técnica formal (Recorridos e inspecciones)***, *la cual es una reunión de personal técnico cuya principal misión es el descubrimiento de problemas de calidad durante el desarrollo, después se aplican* ***pruebas al software*** *con el fin de descubrir errores* mediante varios casos de prueba, con esto se descubren un gran número de inconsistencias en los resultados esperados, sin embargo no son tan efectivas para descubrir todas las clases de errores. Otra actividad de calidad es la ***estandarización*** de procesos y código, y éstos pueden variar de acuerdo al grupo o empresa que los implante; en caso de existir documentación formal de los estándares a seguir, éstos deben ser revisados por el mismo personal de desarrollo ó en otros casos por un grupo especial dedicado al SQA.
La actividad dedicada al ***control de cambios*** es también imprescindible, pues en este mundo tan cambiante, es difícil que los sistemas no queden fuera de este patrón de comportamiento, y más importante aún es el formalizar las peticiones, la naturaleza y los impactos del cambio.
Finalmente los resultados de las revisiones, auditorías, control de cambios, pruebas y demás actividades del SQA deben convertirse en parte del registro histórico de un proyecto y deben ser divulgados a todo el personal de desarrollo que lo solicite(***Registro de información y generación de informes***).

En varias empresas existen grupos especiales para asegurar la calidad de los desarrollos, y éstos pueden ser totalmente independientes del grupo de desarrollo, lo cual suma imparcialidad a cada proyecto y permite que el personal encargado de tal control sea especialista en su ramo, este grupo de SQA tiene como propósito el garantizar que los procedimientos, las herramientas y las técnicas utilizadas en el desarrollo y modificación del producto son los adecuados para alcanzar el nivel de confianza deseado en los productos. Entre las tareas desarrolladas por el personal de control de calidad destacan :

A).- Desarrollo de políticas, practicas y procedimientos estándar
B).- Desarrollo de herramientas de prueba y otros auxiliares para el control de la calidad
C).- Ejecución de las funciones de control de calidad descritas en el plan de control de calidad de software para cada proyecto
D).- Ejecución y documentación de las pruebas de aceptación de cada proyecto.
E).- Conducir el proyecto posterior, pues escribe el legado del proyecto y conserva los registros del proyecto a largo plazo

Las empresa a través de su equipo de SQA ofrece beneficios tales como 1) El software tendrá menos defectos latentes, por lo que se aplicará un menor esfuerzo durante las pruebas y el mantenimiento lo que representa un porcentaje importante en el costo total del desarrollo 2) se tendrá una mayor fiabilidad y por lo tanto un grupo de clientes satisfechos 3) el costo del ciclo de vida del proyecto disminuirá substancialmente.

Por ejemplo, para un grupo de programadores que trabajen en un ambiente WEB podrían tener en mente y como regla general los siguientes 7 puntos, para garantizar que siguen algunos de los estándares establecidos, para alcanzar la máxima calidad :

1. Apariencia general
Es importante mantener un formato que identifique a la página Web de otras de su mismo género, es decir, crear un estilo que se refleje tanto en el contenido y la presentación del sitio y que lo haga diferente y único.

2. Uso de gráficos
El uso de imágenes y logotipos siempre debe estar balanceado de acuerdo a lo estipulado en el punto anterior así como al contenido y carácter general de la página, cuidando siempre el aspecto técnico de no saturarla con demasiada información visual, de lo contrario dificultará su proceso de transmisión de ideas.

3. Navegabilidad del sitio.
Uno de los factores importantes dentro de un sitio Web es la facilidad con que el usuario puede accesar a la información que requiere, de tal manera que siempre sepa en que lugar específico del sitio se encuentra.
Esto se puede lograr mediante una barra de navegación siempre visible así como la creación de un mapa general del sitio.

4. Uso de nuevos productos y tecnología.
Actualizar el sitio Web constantemente con nuevos productos y tecnologías así como su construcción cuidadosa, para proveer una consistente experiencia visual a todos los usuarios, quienes lo visitarán con mayor frecuencia.

5. Calidad de contenido.
Esta es una parte de vital importancia para un sitio Web ya que al usuario le interesa conocer lo que esta recibiendo a través de este medio, siempre presentado de una manera fácil y entendible que

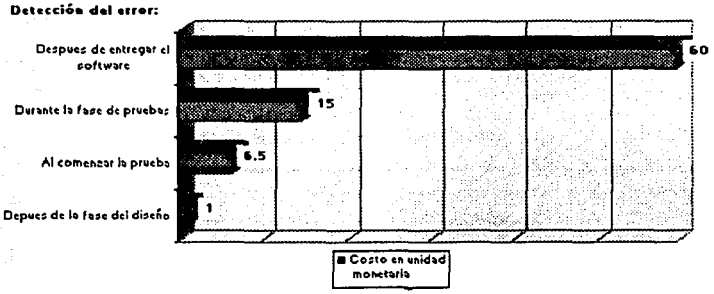
sea para beneficio del usuario y alcance el objetivo esencial del sitio: proveer la información necesaria de manera concisa y eficiente.

6. Eficiencia en la programación
Este factor constituye el soporte del sitio Web ya que de éste depende la funcionalidad general y su fácil acceso desde cualquier plataforma.

7. Mantenimiento del sitio y actualización.
De acuerdo a los puntos anteriores, un sitio Web que ofrece constantemente nuevas alternativas en cuanto a presentación y contenido tendrá una presencia sobresaliente en el medio del Web y resultará en beneficios permanentes en lo que a promoción y comunicación se refiere para la imagen de la empresa

XIV.3.- **Revisiones de Software (Recorridos e Inspecciones)**
Las revisiones técnicas son uno de los filtros más infalibles para descubrir errores ó señalar mejoras al producto, pues a través de terceras personas el proyecto puede vislumbrar errores que el analista de desarrollo paso por desapercibido. La principal ventaja de las revisiones es el descubrimiento inmediato de fallos y aveces el anticiparse a posibles errores, lo cual reduce substancialmente el costo en los siguientes pasos de las fases de desarrollo y mantenimiento.

La siguiente gráfica muestra la importancia económica de la detección de problemas a tiempo :



Los recorridos e inspecciones consisten básicamente en revisiones metódicas del proyecto por parte de un grupo de expertos ó elementos de algún grupo de desarrollo.
Los elementos que se pueden revisar son :

- Los requisitos (Docto. de solicitud)
- Las especificaciones de diseño
- Los planes de prueba
- El código fuente
- Los principios de operación
- Los manuales de usuario
- Los procedimientos para el mantenimiento.

Las revisiones pueden descubrir y resaltar las áreas problema, y estos deberán quedar registrados para que posteriormente sean corregidos. El personal que puede intervenir en la revisión son : el productor de software, un jefe de revisión, 2 o 3 revisores y un redactor técnico, quienes proporcionarán sus diferentes puntos de vista, éstos pueden ser líderes de proyecto o personal del equipo de control de calidad o hasta un representante de otro proyecto. Una revisión debe establecerse en una atmósfera de cordialidad y entendimiento, y no de manera negativa y defensiva, a su vez es recomendable que no asistan directivos o administradores, ya que aveces pueden inhibir el proceso de revisión, las revisiones se deben realizar en base a un calendario de trabajo, deben de quedar registrados los errores en un "*sumario de revisión*" para posteriormente ser corregidos, las sesiones de recorrido no deben de generar grandes análisis en problemas menores, las sesiones deberán durar menos de 2 horas para que de esta forma se ubiquen en los principales acontecimientos e se pueda incentivar la participación de los revisores.

La metodología de la revisión consiste en proporcionar el material que se tiene del proyecto a los revisores con al menos un día de anticipación, y durante la revisión; el productor (Líder de proyecto) comenzará con una breve introducción y explicará el material presentado ; mientras tanto el redactor va anotando los problemas descubiertos. Al final de la revisión los revisores tomarán una decisión de acuerdo a lo siguiente:

- El producto no presenta problemas, por lo que se acepta tal cual
- El producto presenta errores menores, por lo que se acepta y se procede a la inmediata corrección, por lo que ya no será necesaria otra revisión.
- El producto se rechaza debido a serios errores y cuando se encuentre corregido se evaluará de nuevo

El porque de las revisiones :
El tiempo empleado para los recorridos es una inversión que mejora de forma inmediata y durante toda la vida la calidad de proyecto, pues los problemas se detectan cuando su corrección es más fácil y menos costosa, aumenta la comunicación entre los miembros del equipo, y son un excelente medio educacional pues el personal aprende las técnicas de los demás y el personal de nuevo ingreso aprende con rapidez los detalles de los proyectos, además los productos de trabajo se llegan a ver como documentos públicos y existe una mayor motivación y satisfacción del grupo en general.

XIV.4.- Pruebas de Software
En el mundo actual, cada día es mayor el número de personas que utilizan y confían en programas hechos para satisfacer sus necesidades. Con mayor frecuencia y sin darse cuenta las personas y empresas dependen cada día mas de sistemas de información y control en sus trabajos, hogares y hasta en la diversión.
Desgraciadamente no todos los programas hechos para satisfacer estas necesidades son infalibles, y existen errores que han aparecido en programas en infinidad de formas, como los errores de sintaxis, de lógica, de capacidad, y hasta de juicio.
El uso de programas que contengan errores puede causar riesgos en cualquier empresa en donde los gerentes, empleados o médicos se basen en lo que la computadora les han reportado.
Cualquier persona con ligeros conocimientos de programación sabe que un programa puede llegar a fallar por alguna razón, y para cualquier persona común esto no es aceptable, debido a que piensa que los sistemas de información deben ser perfectos; Sin embargo esta problemática la vivimos a diario, ya sea en algún problema que tengamos al retirar dinero de un banco o que el elevador no haga caso a las indicaciones de llevarnos a la planta baja; es decir la mayoría de la gente no tiene idea de lo dependientes que somos de los programas hechos para satisfacer necesidades de la vida diaria.

A continuación se presentan algunas situaciones en donde un error de programa a causado serios problemas:

* En la NASA los programadores no consideraron la rotación de la tierra alrededor del sol en el aterrizaje de la cápsula espacial Géminis V, por lo que éste cayó a 160 KM. de donde se esperaba.

* En el año de 1985 hubo un fallo en el sistema del Banco de Nueva York, por lo que éste corrompió transacciones de valores del gobierno y al final del día el banco había retirado 32000 millones de dólares de mas, por lo que el banco tuvo que pagar 5 millones de dólares en intereses.

* La máquina de Radiación "THERAC 25" para rastrear cáncer en pacientes se utilizó con éxito en miles de personas, sin embargo un error del programa hizo que se aplicara una sobredosis de radiación a dos pacientes, causándole la muerte a uno de ellos y daños irreversibles al otro.

* En Enero de 1990 falló el sistema de comunicación de larga distancia de AT&T, por lo que durante 18 horas estuvo suspendido el servicio ocasionando que 20 millones de llamadas no llegaran a su destino.

*El sistema de reservaciones de un lujoso hotel en Estados Unidos tuvo problemas al asignar una misma habitación a varias personas, por lo que algunas huéspedes molestos llegaron a demandar al hotel.

*En el lapso de un 1 año el sistema controlador de tráfico aéreo del Aeropuerto de Londres se averió 5 veces, a pesar de tener a 70 especialistas encargados del mantenimiento y actualización del software.

Todos estos errores de software fueron detectados en algunos casos en una línea de código mal escrita de entre decenas de miles de ellas, por lo que puede ser sumamente difícil encontrar el error y aún más difícil eliminarlo. La experiencia nos ha enseñado que en Macro-Sistemas o sistemas medianos, generalmente el corregir un problema nos lleva a otros fallas que antes no existían, por lo que esta comprobado que los intentos por corregir errores pueden ocasionar otros errores.

Las pruebas de Software son costosamente necesarias, pues en algunos proyectos se invierte el 40% o más del tiempo de la duración de todo el proyecto, y en ellas se descubren problemas que de no haber sido identificados a tiempo, entonces los errores hubieran ocasionado grandes costos en el servicio o a veces daños irreparables, por lo que las pruebas de software son un elemento crítico para la garantía de calidad de los desarrollos.
Las pruebas demuestran hasta que punto las funciones de software parecen funcionar de acuerdo con las especificaciones y si se pueden alcanzar los requisitos de rendimiento, también a través de los datos obtenidos se puede proporcionar una mayor o menor confianza en el desarrollo, sin embargo el objetivo de la prueba no consiste en que todo marche bien, sino por el contrario, el obtener y descubrir errores no descubiertos hasta entonces.

XIV.5.- Probando unidades de programa

Las pruebas de unidad de programa son aquellas que se aplican a los módulos o subrutinas que se utilizan en cada programa y se prueban de manera minuciosa e independiente antes de integrarse al sistema. Se deben diseñar casos de prueba para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiado y cada caso de prueba debe acompañarse

de un conjunto de datos o resultados esperados. Regularmente son 4 pruebas las que se utilizan para las unidades de programa.

1.- Las pruebas funcionales consisten en incluir datos de entrada para los cuales se conoce el resultado esperado.
2.- Las pruebas de desempeño implican el medir la cantidad de tiempo de ejecución, la eficiencia del programa, y la utilización óptima de dispositivos por la unidad de programa. Dependiendo del grado de tiempo utilizado y requerido, se puede reescribir el código para optimizar los tiempos en donde la unidad del programa es demasiado lenta, sin embargo no hay que perder de vista que se puede desperdiciar mucho tiempo en el ajuste fino de un subprograma y que éste no contribuya en gran medida al desempeño global del sistema.
3.- Las pruebas de tensión consisten en demandar recursos de cantidad, frecuencia ó volúmenes anormales para tratar de generar inestabilidad o procesamientos incorrectos y de esta manera romper con el procesamiento interno del módulo.
4.- Las tres primeras pruebas son conocidas como pruebas de caja negra y existe otro tipo de prueba llamado "Prueba de Estructura" o también conocida como prueba de caja blanca ó de cristal, éstas últimas utilizan la estructura de control del diseño procedimental, para derivar diferentes casos de prueba y tratar de garantizar lo siguiente:

- Que se ejecuten por lo menos una vez, todos los caminos independientes de cada módulo
- Que se ejerciten todas las decisiones lógicas con sus correspondientes valores de falso y verdadero
- Que se ejecuten todos los bucles con sus límites operacionales
- Que se utilicen todas las estructuras internas de datos para aseverar su validez.

Por lo que podemos observar que las pruebas de unidad básicamente se prueban las interfaces del módulo, para asegurar que la información fluye de manera adecuada hacia adentro y hacia afuera de la unidad que esta siendo probada, se examinan las estructuras de datos propias del módulo para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo, se prueban las condiciones extremas para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento, Se ejecutan todas las estructuras de control con el fin de asegurar que todas las sentencias del módulo se ejecutan al menos una vez.

En todo tipo de pruebas de software no se sabe con exactitud si ya han sido suficientes las pruebas realizadas, ya que estando en producción una corrida más por parte de los usuarios representa en sí otra prueba con un nuevo conjunto de datos. No podemos tener la absoluta certeza de que el software nunca fallará, pero en base a reportes estadísticos se puede decir que si se realizan rigurosas pruebas de cada sistema particular, se puede aseverar que el software presenta un 95% de funcionamiento libre de fallos.

Algunas veces se da por hecho que existirán problemas debido a los fallos en Hardware o Software o problemas provocados por los usuarios, por lo que a continuación se presentan algunos de los posibles problemas y causas de un sistema en particular :

Problema no. 1
Descripción : Al intentar entrar a la aplicación, aparece una ventana con la leyenda : "there was no response, the server could be down or is not responding"

Probables causas :
- La máquina del usuario tiene problemas de comunicación
- El puerto se encuentra inactivo
- El servidor no responde

Problema no. 2
Descripción : Al introducir el usuario y el password, el sistema el sistema despliega una ventana como la siguiente :

Usuario sin autorización

Aceptar - Cancelar

y al cancelar aparece una página en blanco como la siguiente :
error 401
Unauthenticated Exception
Lotus-Domino Release- 4.6.4a

Probables causas :
- Se ha introducido incorrectamente la clave de acceso y/o password
- Se encuentra activado/Desactivado el bloque de mayúsculas
- El usuario no se encuentra dado de alta en la base de datos

Acción a realizar :

Problema no. 3
Al visualizar la pantalla para ingresar los datos de una nueva solicitud, en las listas del tipo moneda no aparece la información del catálogo corporativo

Probable causa :
- La base de datos se encuentra inactiva

Problema no. 4
Al abrir la pantalla para ver alguna solicitud, aparece una ventana con la siguiente leyenda :

JAVA_SCRIPT ERROR :
Java_Script function not declared

Probable causa :
- El browser 2.0 no termina de cargar la forma adecuadamente

XIV.5.1- Pruebas de Integración

Una vez que se ha terminado con la programación de todos los módulos constituyentes del sistema, se lleva a cabo la unión y pruebas del sistema completo, es decir, que el sistema esta listo para soportar las pruebas de integración, las cuales se realizan para garantizar que los diferentes módulos interactúen de manera ordenada y correcta, para detectar y corregir perdidas en las interfaces y en las estructuras de datos globales. Existen tres técnicas de integración: Integración ascendente, descendente y de resguardo.

Las **pruebas de integración ascendentes** consisten en revisar cada unidad de programa en forma exhaustiva, posteriormente en revisar los diferentes subsistemas, verificando las interfaces de control y de datos que se manejan en los diferentes módulos y finalmente se prueba el sistema completo, que se relaciona con las sutilezas de las interfaces, la lógica de decisión, el flujo de control, los procedimientos de recuperación, la eficiencia global, la capacidad y las características del sistema entero.

Las **pruebas descendentes** consisten en ejecutar primero el sistema principal, y posteriormente correr las demás rutinas subordinadas, con esto se prueban con mayor frecuencia las rutinas de nivel más alto, los módulos se integran a medida que se desarrollan y los errores se localizan en los nuevos módulos e interfaces que se están añadiendo; sin embargo se presenta cierta dificultad al tratar de probar niveles inferiores y se tenga que pasar por todos los módulos superiores a él, es decir, no es costeable volver a ejecutar "*n*" rutinas cada vez que se agregue un nuevo módulo ; muchas veces puede existir un ahorro de tiempo al probar la rutina por separado antes de ser insertada en la estructura descendente, por lo que a esta combinación de pruebas se le llama **pruebas de emparedado o de resguardo**. La integración descendente puede empezar a aplicarse por profundidad y por anchura y esto dependerá de las características específicas de la aplicación y del diseño detallado.

Una vez realizada la integración del sistema se deben de realizar dos tipos de pruebas por parte del equipo de desarrollo, las pruebas de recuperación y las pruebas de seguridad, las primeras están encaminadas a que todo sistema debe ser tolerante a fallos, es decir, los errores en el procesamiento no deben hacer que se detenga el funcionamiento del sistema, por lo que la prueba de recuperación es una prueba que fuerza al fallo del software de muchas formas y lo más importante, verifica que la recuperación se lleve a cabo apropiadamente y de forma automática. Las pruebas de protección o seguridad se deberán aplicar en todo sistema que maneje información importante o que lleve a cabo acciones que puedan beneficiar o perjudicar a terceras personas. Los sistemas pueden ser intervenidos por empleados disgustados que tratan de penetrar por venganza o por personal que se quiere aprovechar de la situación para obtener una ganancia impropia o por empleados que solo están experimentando su ocio. El objetivo de las pruebas de seguridad es el tratar de accesar a las bases de datos, al código fuente ó el de obtener las claves de acceso para dañar al sistema, y por desgracia con el suficiente tiempo y recursos una buena prueba de seguridad terminará por penetrar en algunos sistemas, sin embargo el rol del programador del sistema para contrarrestar los accesos piratas es el de hacer que sea más costosa la violación del sistema que el valor de la información obtenida.

XIV.5.1- Pruebas de Integridad

El objetivo principal de estas pruebas es el de comprobar que todo dato introducido por el usuario es el correcto para el contexto en el cual se solicita (p. ej. la fecha de matrimonio de una persona no puede ser menor a su fecha de nacimiento). También se revisa que cada botón del sistema realice la opción deseada y no otra diferente (ver anexo 2.5: pruebas de integridad).

XIV.6- Herramientas automáticas de prueba :

Debido a la gran inversión que se tiene que hacer en las pruebas de software, se han desarrollado herramientas que nos ayuden a recortar tiempos en la realización de nuestras pruebas, y se espera que la aplicación de estas herramientas automáticas se acelere durante estos años. a continuación se mencionan algunas de estas herramientas automáticas de prueba :

Verificadores de comportamiento : Identifica en qué áreas se esta empleando mayor cantidad de tiempo con respecto a otros eventos, por lo que posteriormente se debe determinar si en éstas regiones es conveniente optimizar el código fuente.

Verificadores de estándares : Pueden servir para inspeccionar el código y encontrar incongruencias con los estándares preestablecidos y de los principios generales de programación estructurada.

Analizadores estáticos o de cobertura : Éstos registran las rutas de control que siguió cada prueba, para que posteriormente se informe la extensión o recorrido que siguieron todas las pruebas, esto se hace con el fin de identificar el número mínimo de rutas que se deben ejecutar para lograr la cobertura de todas las ramas de decisión en un módulo.

Comparadores de salida : Determina las diferencias obtenidas con la salida de un programa, contra un archivo previamente registrado.

Simuladores de ambiente : Se utilizan para aparentar el entorno de operación en el que funcionará el software y se utilizan básicamente en situaciones en que la operación del ambiente real es impráctica, como el manejo de máquinas inexistentes, simuladores de entrada de tiempo real para un sistema inexistente o de costosa operación y situaciones en donde las pruebas en vivo son imposibles. Es decir se desea poder simular dinámicamente las condiciones de operación ; ejemplos de este tipo de pruebas los podemos ver en simuladores de vuelo comercial y militar y en programas espaciales.

XIV.7.- Pruebas de aceptación o validación

Generalmente cuando se terminan de realizar las pruebas de integración se comienza con el desarrollo de las pruebas de aceptación, las cuales implican la planeación y ejecución de las ya mencionadas pruebas de funcionamiento, de desempeño y de tensión, para saber si el sistema cumple con las especificaciones convenidas. El cliente realiza una primera corrida de las pruebas en un lugar físico y un ambiente de software controlado por el proveedor de software, y los errores encontrados son registrados para su rápida corrección, a este tipo de pruebas bajo este ambiente se le denomina pruebas "alfa", mientras que las pruebas denominadas "beta" las ejecutan distintos usuarios en lugares no controlados por el proveedor. A raíz de las pruebas alfa y beta el cliente puede o no aceptar el sistema de acuerdo a las especificaciones convenidas desde el principio del desarrollo, es decir, si existen aún errores el equipo de desarrollo deberá realizar las modificaciones pertinentes y preparar así una nueva versión del producto. La metodología para realizar este tipo de pruebas lo podemos ver mas a detalle en el anexo 2.4.

XIV.8.- Fase de Liberación

Una vez que se ha demostrado que el software fue desarrollado conforme a las reglas preestablecidas (verificación) y cumple con los requisitos del cliente (validación). El siguiente paso consiste en liberar el sistema a producción. Nuestro producto esta listo para enfrentarse al mundo.
Sin embargo, debe quedar constancia de que tanto el desarrollador como el cliente están de acuerdo en que esto es lo correcto. Para ello es necesario redactar la Carta de Liberación del software, en la cual se especifica que, dado que se han cubierto todos los requisitos del Cliente (Carta de Especificaciones) el software puede empezar a utilizarse.
Esta formalidad tiene como finalidad poner un límite a la fase de desarrollo del software. Toda modificación, todo nuevo requerimiento sobre un programa que ya fue liberado, deberá tratarse como un proyecto nuevo. Ver ejemplo de Carta de liberación en el anexo 2.5 .

XIV.9.- Control de calidad en los mantenimientos

En caso de existir en la empresa un grupo de SQA, éste debe de revisar que la calidad de software no disminuya con las actividades de mantenimiento por lo que debe vigilar las solicitudes de cambios, debe preparar resúmenes e informes de tales solicitudes, debe realizar las pruebas de regresión de las modificaciones del software, debe verificar que la estructura y la documentación interna del código fuente no se destruya con las soluciones rápidas, debe otorgar protección física a los respaldos y a los conjuntos de pruebas, debe realizar un inventario de las versiones y productos de software que se tienen en distintos sitios, debe tener la facultad de autorizar y revocar versiones de software (Puede evitar realizar cambios que beneficien a un solicitante pero que tenga un impacto negativo sobre los demás usuarios o sobre las políticas de la organización); además este grupo tiene que estar presente en la junta de control de cambios y en muchas veces es quien debe administrar la junta.

Los grupos de SQA registran los datos importantes en los mantenimientos, para realizar reportes y verificar realmente cual es el costo y beneficio de los mantenimientos, los datos que generalmente registran para tener un mejor control son los siguientes:

- Nombre del programa
- Número de sentencias fuente
- Lenguaje de programación utilizado
- Fecha de instalación del programa
- Número de ejecuciones del programa desde su instalación
- Número de fallos del programa desde la instalación
- Identificación y tipificación de los fallos
- Nivel e identificación de los cambios sobre el programa
- Número de sentencias fuente añadidas en los cambios del programa
- Número de personas-hora por cambio

- Fecha de cambio del programa
- Registro de la persona que realiza el cambio
- Identificación de la solicitud de cambio
- Tipo de mantenimiento
- Fechas de inicio y término del mantenimiento
- Número de personas - hora acumuladas en el mantenimiento
- Beneficios cuantitativos y cualitativos derivados del mantenimiento

RESUMEN

La ingeniería de software es la herramienta básica con que los ingenieros de desarrollo se pueden apoyar, para tratar de realizar proyectos confiables, útiles, portátiles, eficientes y claros, pues a través de sus técnicas y métodos podemos contribuir a mejorar la calidad general del sistema, desde su fase de análisis hasta la de mantenimiento. Con una Ingeniería de software bien aplicada, podemos minimizar los gastos y maximizar los beneficios que de ella se derivan, pues el costo actual de los desarrollos, representa uno de los gastos mas importantes en casi todas las empresas.

La confiabilidad del software es uno de los principales retos para los ingenieros de desarrollo, pues las técnicas actuales de desarrollo de software no aseguran en un 100% que la aplicación funcionará sin fallas bajo todas las circunstancias o caminos posibles. Conforme cada vez más empresas dependan de los sistemas de computación, cada vez será mas importante que los ingenieros encuentren formas para crear software en el cual podamos confiar nuestra salud, nuestro dinero, nuestra diversión, etc. La SQA es la encargada de garantizar que el software programado tenga un grado de confiabilidad del 99.99% como mínimo para poder ser liberado y en algunas empresas existen áreas de calidad con métodos y estructuras rígidas, para que la calidad se haga valer.

Es de suma importancia considerar el buen funcionamiento de cada sistema, apoyados en base a la ingeniería de software, por lo que podemos aplicar 5 documentos básicos para que la calidad del producto de software pueda verse beneficiada :

1. Solicitud de Desarrollo
2. Carta de especificaciones
3. Pruebas de Integridad
4. Pruebas de Aceptación o validación
5. Carta de liberación

Capítulo
TRES

PLANEACIÓN Y ADMINISTRACIÓN DE UN PROYECTO DE DESARROLLO

El objetivo de la planificación es el de proporcionar información al personal directivo, que le permita conocer estimaciones aproximadas de tiempos, recursos y costos del proyecto. Los pasos generales que se deben considerar en la planificación de un proyecto de programación se sugieren en la siguiente tabla.

Definir y justificar el problema - Describir el problema mediante la inclusión de la situación actual, las restricciones existentes y las metas que se desean lograr, así como justificar la necesidad de automatizar el proceso actual Considerar varias estrategias de solución. - Elaborar varias estrategias de solución sin contemplar restricciones y posteriormente determinar cual es la mejor en base a los recursos disponibles y a las políticas de la empresa Establecer estimaciones - Determinar los costos preliminares del proyecto (horas hombre, esfuerzo dedicado, etc.) Determinar el camino de desarrollo - Definir el ciclo de vida y la estructura que tendrá el proyecto - Establecer las herramientas y notaciones a utilizar - Preparar planes de prueba para cada módulo y para todo el proyecto

Como primer paso se debe determinar problema y esto se hace con la obtención de un enunciado breve que identifique o defina al problema, éste debe ser entendible en los términos que utiliza el cliente y además se deberá incluir una pequeña introducción en donde se describa la situación actual del entorno, así como los alcances que deberá tener el nuevo sistema.

Para conocer a fondo e identificar plenamente el problema el planificador debe sentarse a platicar con su cliente, observar la operación o desarrollo de la parte a resolver y tratar de ver otros puntos de vista con las personas que trabajan alrededor, para tratar de interpretar los distintos escenarios y formular de manera precisa lo que el cliente realmente quiere.

Conociendo las capacidades y carencias de los sistemas de desarrollo se debe llegar a determinar si en verdad es necesario una solución computarizada, ya que aveces la solución consiste en comprar un sistema comercial, realizar un sistema propio y a la medida, o aveces ni siquiera es posible el uso de las técnicas informáticas; además se debe determinar si el sistema es costeable económica, social y políticamente.

XV.- FACTORES Y HERRAMIENTAS DE PLANEACIÓN

XV.1.- Establecimiento de metas y requisitos
Para cada proyecto se deben establecer metas cualitativas y cuantitativas, para que estas sean el marco de referencia a seguir, es decir, son los objetivos a los cuales el sistema finalmente tendrá que alcanzar y se debe considerar lo siguiente:

1.- Nuevas capacidades por proporcionarse
2.- Previas capacidades para preservar/mejorar
3.- Nivel de complejidad del usuario
4.- Requisitos de eficiencia
5.- Requisitos de confiabilidad
6.- Modificaciones factibles
7.- Prioridades de instrumentación y condiciones iniciales
8.- Requisitos de portabilidad
9.- Temas de seguridad

Los requisitos determinan las capacidades con las cuales debe operar el sistema, para poder dar solución al problema, además de proporcionar estándares de desarrollo y de control de calidad tanto para el desarrollo, como para el producto; también se establecen para tener un mejor control de la funcionalidad, del equipo y su rendimiento, del tipo de software a utilizar, de las interfaces con el usuario, etc. Se debe realizar un verdadero esfuerzo en determinar los requisitos, ya que aveces es difícil cuantificarlos en la fase de planeación, debido a que aún no está claro lo que se necesita para resolver el problema.

Los planes describen las acciones a realizar para alcanzar las metas y requisitos propuestos y sirven para alcanzar los logros del proyecto a tiempo; se puede entender por logro a aquellas actividades que sirven para alcanzar un objetivo. Para planear el modo de alcanzar los logros se tienen que desglosar y asignarles tiempos de entrega y recursos apropiados, un ejemplo de este tipo de planeación son las cascadas de compromiso.

XV.2.- Cascadas de compromiso
Las cascadas de compromiso son diagramas que representan las actividades a realizar, informan sobre el personal involucrado para el desarrollo de las actividades y determinan los tiempos de cada tarea; es decir en ellas tenemos el control visual del proyecto para saber "que, quien y cuando" realiza determinada actividad.

Cascadas de compromiso:
Es la herramienta de seguimiento que se ocupa para garantizar el cumplimiento de las tareas en cada proyecto. En ella se presentan preguntas como:

- Por qué del proyecto ?
- Quienes lo van a hacer ?
- Cuando lo van a hacer ?
- Que pasa si no lo hacen ?

Los elementos que contiene esta cascado son :
Título : Es el nombre del proceso o conjunto de procesos que están involucrados en el proyecto
La fecha de actualización : Indica la última revisión (Calificación al cumplimiento de los compromisos) que se haya hecho.
El objetivo : Es el enunciado que da sentido al porque del proyecto.
El impacto : Es la descripción cualitativa y cuantitativa de lo que vale alcanzar el objetivo, en términos de servicio, costo, competitividad y rentabilidad (en ese orden).
Los responsables : Son las iniciales del nombre y apellido o algún mnemónico que identifique a cada participante de las personas comprometidas en alcanzar algún objetivo parcial de alguna etapa. También se tiene que incluir al líder del proyecto y/o dueño del proceso.

Los costos de no calidad : Son los costos asociados de no cumplimiento por algún compromiso, en términos de servicio (Tiempo, Imagen, costo, etc.).

Las etapas o conceptos : La cuales engloban las actividades homogéneas y conforman cronológicamente o en otro orden (dependiendo de la naturaleza del proceso)el o los procesos involucrados en el proyecto
En cada etapa se tiene una **fecha objetivo** de cumplimiento de los compromisos involucrados, un **objetivo, política o lineamiento a seguir** y la descripción de cada compromiso.
Cada **compromiso** se compone por el enunciado breve de la actividad o conjunto de actividades a realizar, las **iniciales** o claves que identifiquen a las personas involucradas y la **fecha comprometida**, subrayando el o los responsables directos del cumplimiento, en caso de tratarse de varias personas que conformen un equipo de trabajo.

Al iniciar la cascada, todas las etapas se presentan en blanco, y conforme se va avanzando en las demás etapas, éstas se irán sombreando en color gris si verdaderamente se cumplieron en la fecha indicada y se les pondrá un recuadro negro en caso de que el cumplimiento se halla efectuado antes de la fecha pactada ; y cuando una actividad no se cumplió en la fecha esperada o con la calidad requerida , entonces el cuadro de la etapa se dibujará en color negro.

TITULO DEL PROYECTO

PROCESO QUE INCLUYE

FECHA DE ACTUALIZACIÓN

OBJETIVO:

Enunciado de las variables criticas externas e internas de cara a las necesidades de nuestros clientes externos e internos

IMPACTO:

Poner el valor del proyecto en impacto en mejora del servicio (tiempo de respuesta) y/o ahorro en costos cuantitativo y cualitativo

FECHA OBJETIVO

Objetivo, politica o meta a cumplir en cada etapa

- Compromiso 1 Pendiente/ en Proceso ABC,DEF, HIJ (fecha actual o futura)
- Compromiso 2 cumplido ABC,DEF, HIJ (fecha actual o pasada)
- **Compromiso 3 incumplido ABC.DEF. HIJ (fecha pasada)** → 1
- Compromiso 4 cumplido antes ABC,DEF, HIJ (fecha futura)

→ Etapa 1 o concepto 1 que engloba actividades homogeneas

FECHA OBJETIVO

Objetivo, politica o meta a cumplir en cada etapa

- Compromiso 5 Pendiente/ en Proceso ABC,DEF, HIJ (fecha actual o futura)
- Compromiso 6 cumplido ABC,DEF, HIJ (fecha actual o pasada)
- **Compromiso 7 incumplido ABC.DEF. HIJ (fecha pasada)** → 2
- Compromiso 8 cumplido antes ABC,DEF, HIJ (fecha futura)

→ Etapa 2o concepto 2 que engloba actividades homogeneas

RESPONSABLES:

Dueño del proceso	
(iniciales) - ABC - DEF - GHI - JKL	Nombre y apellido 1 Nombre y apellido 2 Nombre y apellido 3 Nombre y apellido 4

COSTOS DE NO CALIDAD:

		Atraso en la entrega al cliente int.	Atraso en la entrega al cliente ext.	Impacto al clientet.	Costo incurrido.
1	compromiso incumplido	dias, horas, etc.	dias, horas, etc.	servicio, calidad	$
2	compromiso incumplido	dias, horas, etc.	dias, horas, etc.	servicio, calidad	$

ESTADO DE CUENTA CERTIFICADO HIPOTECARIO

PROGRAMACIÓN DE PANTALLAS DE CAPTURA, GUARDAR Y RECUPERAR INFORMACIÓN, GENERAR CÁLCULOS DEL ESTADO DE CUENTA, GENERAR REPORTE FINAL Y PRUEBAS.

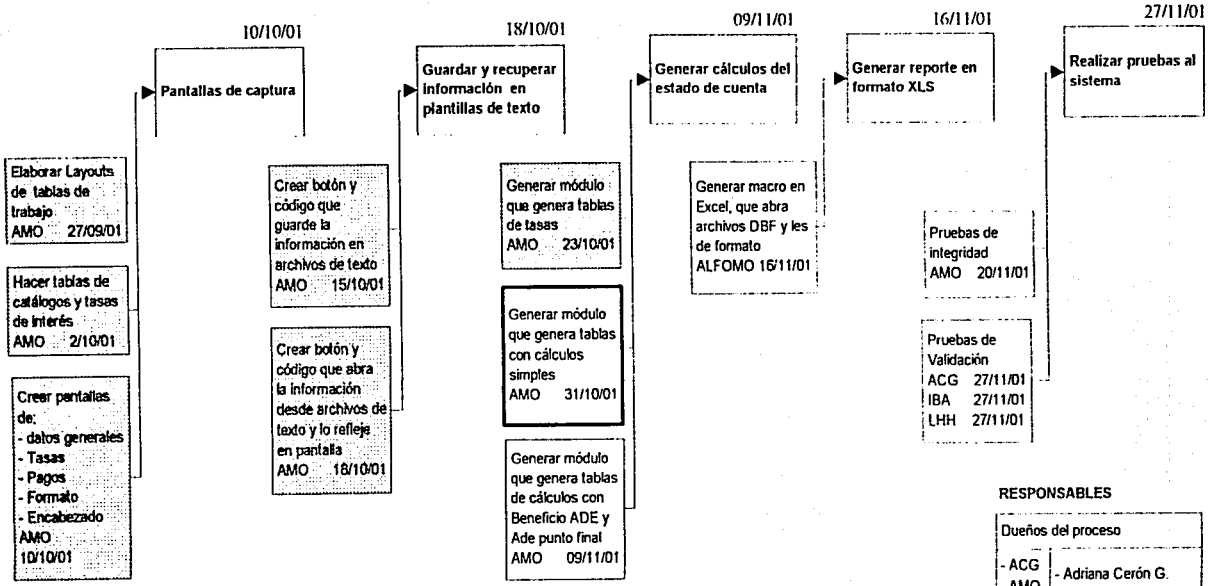
04 DE OCTUBRE DE 2001

OBJETIVO:

Automatizar los Estados de Cuenta Certificados del esquema Hipotecario, modalidad (Interés Social)

IMPACTO:

- Recuperar cartera vencida más rápido
- Evitar errores y reprocesos en la elaboración manual
- Permitir la centralización de este servicio
- Ahorrar 40 hrs. hombre al mes



RESPONSABLES

Dueños del proceso	
- ACG - AMO - IBA - LHH	- Adriana Cerón G. - Alfonso Martínez O. - Ivonne Bernal A. - Lidia Hernandez H.

XV.3.- Desarrollo de la estrategia de una solución

Se deben plantear varias estrategias de solución a través de una tormenta de ideas, esto con el fin de tomar distintos puntos de vista y no encajonamos en una sola idea, estas soluciones deben proponerse tomando en cuenta los factores estratégicos que se desean en el producto final. posteriormente se elige una o mas soluciones para poder realizar estudios de **factibilidad y estimaciones** de costos preliminares y con esto poder ver si las metas y requisitos del proyecto se pueden satisfacer dentro de las restricciones de tiempo disponible, recursos y tecnología de la empresa.

Ya que se ha optado por seguir una estrategia de solución, entonces ésta se debe documentar para servir como justificación a revisiones futuras y no se debe olvidar el incluir la razón del rechazo de otras soluciones.

Ejemplo de una estrategia de solución para resolver la alimentación de información en un sistema personalizado

- Antecedentes

Dentro de una institución crediticia existe un departamento que elabora estados de cuenta de tarjetas de crédito (TDC) de manera automática, por lo que actualmente se corren programas de macros, que obtienen información directa del sistema que administra dichas tarjetas ; Una vez recabada la información se procede a generar automáticamente los estados de cuenta; Sin embargo en poco tiempo, las macros que obtienen la información necesaria para la elaboración automatica ya no podrán interactuar con el sistema, debido a un cambio en el ambiente del sistema administrador de tarjetas de crédito (se cambia la plataforma UNISYS a plataforma DB2), por lo que ya no se podrá obtener información directa de dicho sistema, y con esto, tampoco se podrán elaborar los estados de cuenta de TDC automáticamente.

Problema: Ya no se cuenta con la información necesaria para elaborar automáticamente estados de cuenta de TDC

- Posibles soluciones:

- Solución 1: Conectamos directamente a la base de datos que administra dichos créditos, para que de esta forma realicemos las consultas cuando sea necesario.
- Solución 2: Que el área de "bases de datos" proporcione los archivos necesarios, de tal manera que se puedan realizar consultas y extracción de información desde el área que desea la explotación de la información.
- Solución 3: Que se alimenten manualmente las condiciones necesarias, para que posteriormente se ejecute el proceso automático de elaboración de estados de cuenta de TDC.
- Solución 4: Entregar al área de bases de datos un listado de los números de tarjetas por elaborar su estado de cuenta, y que ésta área nos entregue solo los campos que se requieren para la elaboración de los certificados, es decir sustituir la macro que obtenía información, por un requerimiento periódico al área de bases de datos para que nos entregue esa misma información y poder realizar posteriormente los certificados de forma manual.

- Análisis de soluciones : Después de investigar cada una de las soluciones anteriores se determino lo siguiente :

- Análisis de Solución 1: El área de bases de datos no esta dispuesta a que afectemos su información en línea, debido a que ellos también realizan varias consultas y sobre las cuales afectaremos la velocidad de respuesta de sus procesos actuales.
- Análisis de Solución 2: Los archivos que proporcionará el área de bases de datos tendrán que refrescarse periódicamente, debido a que la información tiende a crecer y además puede cambiar, además dicho traspaso de archivos implica tener almacenados grandes volúmenes de información, por lo que se necesitarán equipos de cómputo con gran capacidad.
- Análisis de Solución 3: Alimentar manualmente las condiciones iniciales para la generación de "n" estados de cuenta certificados implica el Ingreso de 30% de errores por captura, así como un incremento del 1500% en el tiempo de captura.
- Análisis de Solución 4: El área de bases de datos tiene la capacidad de responder a este requerimiento, además de controlar y disparar la búsqueda de información cuando ellos lo crean conveniente, además podrán ejecutar el proceso cuando otros procesos no se estén ejecutando o a horas no pico.

Conclusión de soluciones :

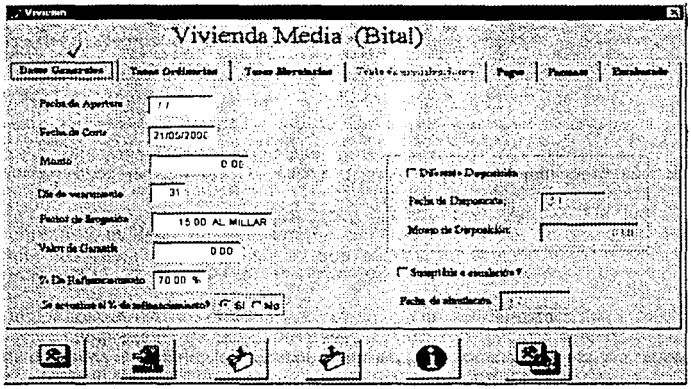
Podemos notar que las soluciones 1 a la 3 con un poco de más de esfuerzo y tiempo se podrían llevar a cabo con algunos contratiempos en su ejecución. Y es notorio que la opción no. 4 presenta todas las facilidades para su realización, pues con un menor esfuerzo y tiempo podemos lograr nuestro objetivo.

XV.4.- Determinación de Prototipos

Un prototipo es una representación o modelo del producto de desarrollo e incorpora algunos componentes del producto real, no llegando estos a ser 100% confiables y con un funcionamiento limitado, sin embargo sirve para tener una idea más clara de los que será nuestro sistema ya terminado, es decir, nos muestra los formatos de entradas de datos, cajas de diálogo, y algunos informes hacia el cliente, por lo que es la aproximación a la versión no. 1 del sistema por desarrollar.

Los prototipos algunas veces son utilizados en la fase de planeación y se utilizan para obtener una mejor comprensión de las necesidades del usuario, ya que simulan aspectos técnicos, simulan despliegues al usuario, formatos de informes, cajas de diálogo, etc. sin embargo éstos suelen tener una funcionalidad limitada, poca confiabilidad y caracteristicas operativas muy pobres.

En la siguiente pantalla prototipo de captura, se puede mostrar al cliente los campos que contendrá dicha pantalla, así como la distribución y validación de cada campo



Descripción de pantalla prototipo :

CAMPO	DESCRIPCIÓN	ENTRADA	VALIDACIONES
Fecha de apertura	Fecha inicio de cálculos del edo. de cuenta	dd/mm/aaaa	Se captura de la solicitud y se verifica contra el sistema que los administra
Fecha de corte	Fecha hasta la que se genera el edo. de cuenta	dd/mm/aaaa	El sistema x defaul recomienda el día de elaboración y debe ser menor a la fecha de apertura
Monto	Importe original del estado de cuenta	999,999,999.99	Se captura de la solicitud y se verifica contra el sistema que los administra
Dia de corte	Es el corte que se realizará cada mes	99	mayor a cero y menor a 31
Valor de garantia	Es el valor de la garantia que se deja en prenda, para respaldar el crédito	999,999,999.99	Debe ser mayor al importe original del crédito
% de refinanciamiento	Indica el monto a refinanciar (En %) con respecto al importe original	99	El sistema x default recomienda el 70%y éste no debe exceder del 200%
Se actualiza el % de refinanciamiento ?	El % se puede incrementar de acuerdo al SMM cada año	(SI/NO)	
.	.	.	.

XV.4.- Tipos de Recursos

Los recursos de (Hardware, Software y Humanos) son la siguiente actividad a determinar para la correcta planificación del proyecto. Para Determinar los recursos humanos se deben tomar en cuenta las habilidades y conocimientos técnicos requeridos, así como el numero de personas necesarias para el proyecto ; en este último punto el planificador se puede apoyar en el estudio de la estimación del esfuerzo. Para los recursos de Software el Gestor del proyecto puede ocupar un sin número de aplicaciones, sin embargo deberá utilizar la que mejor le convenga, es decir, la que se ajuste a sus entradas y que a su vez obtenga las salidas deseadas.

XV.4.1- Recursos de Software

- Herramientas de planificación de sistemas comerciales
- Herramientas de gestión de proyectos
- Herramientas de soporte
- Herramientas de análisis y diseño, de programación
- Herramientas de prueba e integración
- Herramientas de simulación y creación de prototipos
- Herramientas de mantenimiento y de estructura.

XV.4.2.- Recursos Hardware

Se deben definir 2 categorías de hardware al iniciar el proyecto, el primero es definir el tipo de máquina en donde residirá la aplicación, el segundo es determinar en donde se desarrollarán las aplicaciones, cuyas características deben ser óptimas para el desarrollo de programas (veloces y con gran capacidad de almacenamiento) y el tercero es el especificar cada elemento de hardware extra que se utilizará para el correcto funcionamiento del proyecto.

XV.4.3.- Recursos reutilizables

Los recursos reutilizables de Software son una fuente que nos puede servir en el desarrollo del proyecto pues no evitarán varios días o meses de programación, sin embargo si éste es demasiado complejo y requiere de algún mantenimiento, entonces el costo de modificación existente puede a veces ser mayor que el costo en desarrollar un sistema parecido.

XVI.- PLANEACIÓN INICIAL DE UN PROYECTO DE SOFTWARE

Muchas veces la fecha final para el término de un proyecto de Software se define irrevocablemente en base a las necesidades de la empresa , y algunas veces la fecha para el lanzamiento del nuevo producto se determina a través de un cuidadoso análisis del elemento del Software. Siendo de una forma o de otra la definición de terminación del proyecto, se debe hacer corresponder el tiempo cronológico con el esfuerzo humano, es decir, se deben distribuir los esfuerzos dentro del marco de referencia a través de una agenda que pueda dar el seguimiento al progreso del proyecto. también para asegurar que el proceso sea eficiente se pueden realizar tareas en paralelo, es decir, después de haber realizado el análisis, la especificación y la revisión de los requisitos entonces ya se pueden empezar a agrupar las actividades de diseño detallado, de codificación y las pruebas de unidad, de tal forma que se lleve a cabo un desarrollo en paralelo. Debido a que las tareas en paralelo suceden de forma asíncrona, el Ingeniero de Software debe determinar las dependencias entre tareas para asegurar el progreso continuo hasta la terminación.

XVI.1.- Puntos estratégicos para el desarrollo de proyectos (Metas, documentos y revisiones).
El establecimiento de metas , puntos de verificación, documentos estandarizados y puntos de control administrativo permiten ver más claramente el desarrollo del producto pues el proceso de desarrollo se percibe mejor y el producto se torna más tangible, lo cual provoca mejoras en la calidad del desarrollo e incrementos en la productividad de los programadores. Existen técnicas que nos ayudan a tener un mejor control sobre el proyecto, las cuales desarrollan una descripción de la red de tareas de todo el proyecto mediante gráficas o tabulaciones; dicha red se define creando una lista de todas las tareas asociadas con el proyecto, lista que aveces se denomina como estructura de descomposición de trabajos.

Existen 2 técnicas: La técnica de evaluación y revisión de programas (Program Evaluation and Review Technique. "PERT") y el método del camino crítico (Critical Path Method "CPM"), las cuales cuales proporcionan herramientas cuantitativas las cuales permiten:

1).- Determinar el camino crítico
2).- Establecer las estimaciones de tiempo más probables para las tareas individuales con la aplicación de modelos estadísticos
3).- Calcular los límites de tiempo para cada tarea individual
4).- Determinar el número de recursos que se utilizarán para cada tarea.

Independientemente de las técnicas de evaluación se sugieren tomar otras medidas para un mejor control del proyecto, como:

- Realizar reuniones periódicas sobre el estado del proyecto, en las cuales cada miembro del equipo deberá dar un informe sobre los progresos y los problemas.
- Evaluar los resultados de todas las revisiones realizadas en todo el proceso de la Ingeniería de Software
- Determinar si los puntos críticos se han alcanzado en las fechas programadas
- Comparar la fecha de comienzo real con la fecha de comienzo planeada para cada tarea del proyecto
- Mantener reuniones informales con los técnicos para conocer subjetividades en el proceso y lo posibles problemas venideros.

Los gestores del proyecto utilizan estos controles para administrar los recursos del proyecto, para hacer frente a los problemas y para dirigir al personal del proyecto, pues cuando se presenta algún percance se deben de tomar las medidas inmediatas como el de disponer de recursos adicionales, el organizar al, personal o hasta redefinir la agenda del proyecto.

Los siguientes puntos generalmente se utilizan en el desarrollo de productos de programación y **muestran documentos característicos, logros del proyecto, revisiones y puntos de control**, sin embargo no todas las actividades y controles presentados a continuación se necesitan para el desarrollo de un sistema, sino solamente se trata de dar un marco general de referencia.

A).- Se comienza con la **Definición del Producto** y con el plan del proyecto, en donde éste último proporciona información de costos y la agenda que lavará nuestro proyecto a desarrollar. El plan de proyecto "no" tiene que ser un documento extenso ni complejo, su propósito es el de ayudar al establecimiento del esfuerzo de desarrollo de software, el plan debe comunicar el ámbito y los recursos a los coordinadores del Software, al personal técnico y al cliente, el plan debe definir los riesgos, debe definir la agenda, los costos y debe proporcionar un enfoque general del desarrollo del proyecto para toda la gente involucrada.

Formato de la definición del sistema:
1.- Definición del problema
2.- Justificación del sistema
3.- Metas del sistema y del proyecto
4.- restricciones del sistema y del proyecto
5.- Funciones que se proporcionarán (equipo / programación / personal)
6.- Características del usuario
7.- Ambientes de desarrollo / operación / mantenimiento
8.- Estrategia de solución
9.- Prioridades para las características del sistema
10.- Criterios de aceptación del sistema
11.- Fuentes de información
12.- Glosario de términos

Formato del plan del proyecto:
1.- Objetivo y alcance del proyecto
2.- Estructura organizacional y técnicas de gestión
(Estructura de admón., de equipos, de distribución del trabajo)
3.- Requisitos preliminares de personal y recursos
4.- Programación preliminar del desarrollo (Diagramas de Gantt)
5.- Estimado preliminar de costos
.- Riesgos del proyecto
6.- Mecanismos de supervisión y control del proyecto
7.- Herramientas y técnicas que se emplearán
8.- Lenguajes de programación
9.- Requisitos de prueba
10.- Documentos de apoyo necesarios
11.- Formas de demostración y entrega
12.- Programación de entrenamiento y materiales
13.- Plan de instalación
14.- Consideraciones de mantenimiento
15.- Método y tiempo de entrega final
16.- Fuentes de información

Y en base a lo anterior se realiza un estudio de factibilidad para ver si el proyecto continúa, se detiene ó se cambia el rumbo del mismo, si fuera así se modificaría la definición del producto y el plan del proyecto, por lo que se tendría que realizar otra revisión de factibilidad

B).- Se prepara un **Manual de Usuario** preliminar, utilizando los resultados de los estudios de prototipos y pruebas y utilizando información de la definición del sistema. Este manual preliminar servirá de enlace entre el equipo de desarrollo y los usuarios y sus puntos importantes se muestran a continuación:

Manual de usuario:
1.- Introducción
2.- Panorama y exposición del producto
3.- Terminología y características básicas
4.- Resumen de informes y despliegues
5.- Arranque
6.- Modos de ayuda
7.- Modos de operación (Comandos, diálogos, informes)
8.- características especiales

C).- Se realiza la **Especificación de Requisitos** en donde se detalla cada requerimiento, así como las interfaces externas hacia el equipo, otros programas y hacia el personal. Cada requisito se debe definir de tal forma que pueda ser probado o inspeccionado. A continuación se presenta un bosquejo de la especificación de requisitos:

Especificación de Requisitos:
1.- Panorama y resumen del producto
2.- Ambientes de desarrollo, operación y mantenimiento
3.- Interfaces y flujo de datos
4.- Especificaciones funcionales
5.- Requisitos de operación
6.- Manejo de excepciones
7.- Prioridades de instrumentación
8.- Criterios de aceptación
9.- Pruebas funcionales y de operación
10.- Estándares de documentación
11.- Guías de diseño
12.- Glosario de términos

D).- Se establece un **Plan de Verificación del Software**, el cual determina lo métodos que se usarán para la verificación de los requisitos definidos en el plan de especificación para la producción de software y se detalla en el siguiente cuadro:

Plan de verificación de software:
1.- Requisitos a verificar
2.- Plan de verificación del diseño
3.- plan de pruebas del código fuente
4.- Criterios de terminación de pruebas
5.- Plan de verificación de documentos
6.- Herramientas y técnicas a utilizarse

E).- Se realiza una **Revisión Global** y detallada de los puntos 1 al 4, para verificar y asegurar la consistencia entre todos los puntos, las metas de esta verificación es que todos estén de acuerdo con la terminología empleada, con la interpretación uniforme de las especificaciones y la exposición de áreas de problemas; una vez estando de acuerdo en las características principales del producto

se elabora un documento formal entre el cliente y el equipo de software, el cual puede servir como contrato en el desarrollo del producto, y cualquier cambio posterior a éste se debe solicitar por escrito y ser aprobado por el equipo de desarrollo o por el cliente, según el caso. Los participantes de esta revisión deberán ser el equipo de análisis y planeación, los representantes del cliente y los equipos de desarrollo y de control de calidad.

F).- El equipo de análisis y/o diseño elabora la **Especificación del Diseño de Software** a través del diseño estructural y detallado y lo hacen en base al acuerdo contractual del paso 5.

Contenido de la especificación del diseño :
- Estructural
1.- Diagramas de flujo de datos del producto
2.- Descripción conceptual de estructuras y bases de datos
3.- Nombres, unidades y otros atributos de los elementos de datos
4.- Nombre y descripción funcional de cada módulo
5.- Especificación de interfaces de cada módulo
6.- Interconexiones entre módulos y estructuras de datos
7 - Restricciones de tiempo
8 - Condiciones de excepción
- Detallado (Descripción física de estructuras y bases de datos)
9 - Especificación de diccionario para todos los elementos de datos
10 - Algoritmos detallados para cada uno de los módulos
11.- Adaptaciones necesarias para el código existente que será reutilizado
12.- Técnicas específicas de programación para resolver problemas especiales
13.- procedimientos de inicio

G).- Se realiza una revisión del Diseño Preliminar, para evaluar su concordancia con la especificación de requisitos, en caso de ser correcto el administrador del proyecto lo autoriza, de lo contrario se rediseña el punto 6.

H).- Se prepara el **Plan de Prueba de Aceptación** , para incluir lo métodos que se usarán para verificar que el diseño y el código fuente sean consistentes y completos con respecto a los requisitos y el diseño. Formato del plan de prueba de aceptación:

Plan de prueba de aceptación
1.- Requisitos a verificarse
2.- Casos de prueba para cada requisito
3.- Resultado esperado para cada caso de prueba
4.- Capacidades demostradas para cada prueba

I).- La Verificación del Sistema de Programación se hace en base el plan de prueba de aceptación, en donde se verifican los casos de pruebas reales, resultados esperados y capacidades que demostrará cada caso.

J).- En la Fase de Desarrollo se escribe y depura código fuente, se prueban por separado y en global sus módulos y los programadores siguen ciertos estándares como son : el estilo de codificación, estructura lógica, definición de datos, inclusión de comentarios, métodos de depuración, etc.

K).- Antes de integrar los módulos de programación, debe existir al menos un revisor del código fuente que avale lo programado.

L).- Durante la evolución del producto se pueden realizar auditorias de calidad, en donde se realicen inspecciones y reccorridos para verificar la integridad, consistencia y adecuación del producto.

M).- Los últimos puntos del desarrollo son la elaboración del manual de usuario, los planes de instalación, la capacitación al personal y el plan de mantenimiento.
N).- Se efectúa una revisión final antes de entregar el producto, para verificar que todos los documentos estén completos y acordes a lo estipulado, por lo cual se prepara un **Resumen de Verificación** con todas las auditorías, inspecciones y pruebas realizadas durante el ciclo de desarrollo.

O).- El Legado del Proyecto es el último documento en la fase final de los desarrollos de software en donde se registran los aciertos y errores cometidos, las lecciones aprendidas y las recomendaciones para futuros proyectos.

Legado del proyecto
1.- Descripción del proyecto
2.- Expectativas iniciales
3.- Situación actual del proyecto
4.- Áreas que falta atender
5.- Registro de actividades/tiempo
6.- Lecciones técnicas aprendidas
7.- Lecciones administrativas aprendidas
8 - Recomendaciones para proyectos futuros

XVI.2.- División del proyecto en estructuras de trabajo.

XVI.2.1.- Estructura Funcional :
En este tipo de estructura se realiza una actividad o función por cada equipo de trabajo, una vez terminada ésta se rota a otro equipo de trabajo para continúe con la siguiente actividad ; una estructura común generalmente la componen tres equipos, uno de análisis, uno de diseño/desarrollo y otro de pruebas/mantenimiento, aunque estas actividades se pueden agrupar de diferente forma. Para este tipo de estructura se requiere una comunicación constante entre los equipos y es recomendable que los integrantes roten entre los equipos para evitar el tedio y la especialización.

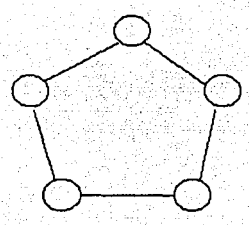
XVI.2.2.- Estructura por proyecto
En este tipo de estructura el equipo de trabajo se dedica de principio a fin al desarrollo de todo el proyecto (Definición, Diseño, Desarrollo, Pruebas, mantenimientos y documentaciones) , y solo en las últimas actividades algunos integrantes pueden ir a nuevos proyectos, sin dejar la responsabilidad de realizar los mantenimientos requeridos

XVI.2.3.- Estructura Matricial (Híbrida)
En esta estructura existe una combinación de actividades por proyecto y funcionales, ya que existen grupos de trabajo que pertenecen organizacionalmente a una función en específico, pero que tienen uno o dos administradores, los cuales son miembros de la planeación y/o desarrollo y pueden participar en la revisión de el diseño, en la elaboración de pruebas y en la creación de manuales.

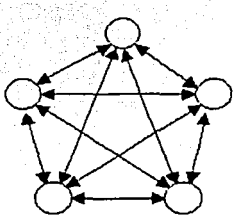
XVI.3.- Estructura del grupo de programadores
Cada proyecto debe tener una estructura interna de comunicación y su formación dependerá de la naturaleza del proyecto. La formación por **Grupo Democrático** es recomendada en proyectos de investigación y en desarrollos largos y difíciles y consiste en que todos los elementos del equipo participan abiertamente en la toma de decisiones y el liderazgo se rota de un elemento a otro dependiendo de las actividades que se realicen y de las capacidades de los integrantes. Las ventajas de trabajar en un grupo democrático son la contribución de tomar decisiones en grupo, del

auto-aprendizaje en equipo y la tranquilidad de trabajar en un ambiente democrático y sin presiones, sin embargo también se presentan ciertas desventajas como la excesiva comunicación entre todos los miembros del equipo, el requisito de que todos lo miembros trabajen juntos y algunas veces se ve reflejada la falta de autoridad y responsabilidad. Existe también la formación con **jefe de programación**, el cual lleva a cabo el control y administración del proyecto, y designa responsabilidades a cada uno de los elementos, este individuo tiene la autoridad y responsabilidad de entregar los productos en tiempos y fechas establecidos y de acuerdo a las especificaciones convenidas para cada desarrollo ; las ventajas de este grupo son la toma de decisiones centralizadas y la reducción en las trayectorias de comunicación, sin embargo su eficacia puede verse afectada por las capacidades técnicas y administrativas del jefe, lo cual puede desmotivar a los miembros subordinados. Una estructura que combina una formación por grupo democrático y con jefe de programación se le llama **estructura bajo jerarquía administrativa**, la cual es útil para el desarrollo de productos de programación modulares, pues cada subsistema se puede asignar a equipos diferentes, y estos pueden funcionar bajo un jefe y mantener una comunicación directa entre cada programador del equipo ; la desventaja que presenta esta estructura consiste en que los mejores programadores suelen ser promovidos hacia posiciones administrativas y no poder desarrollar su nuevo puesto con las habilidades necesarias, es decir que tiene un doble efecto negativo, pues se pierde un buen programador y se crea un mal administrador.

Estructura y trayectorias de comunicación en un grupo sin egoísmo :



1) Estructura



2) Trayectorias de comunicación

XVII.- ESTIMACIÓN DEL PROYECTO

La estimación de cada proyecto es de suma importancia para aproximarnos a conocer cuanto tiempo durará y cuanto personal se ocupará (Recursos) en el desarrollo del proyecto, así como el Hardware y Software que se van a requerir y el riesgo implicado. Para obtener una buena estimación es necesario tener experiencia, contar con buenos datos históricos y la combinación y reconciliación de varias técnicas de estimación (La descomposición, la modelización y las herramientas automatizadas). La complejidad del proyecto es un punto importante para determinar la duración del proyecto y está ligada a la familiaridad y experiencia que el equipo de desarrollo tenga para manejar los requerimientos descritos en el proyecto, el tamaño del proyecto es otro

factor que afecta la precisión y eficacia de las estimaciones, pues a medida que el proyecto se descompone, aún así los módulos pueden ser grandes y complejos.

La estimación se realiza durante la fase de planeación y ésta resulta tan solo preliminar, debido a que no es posible realizar estimaciones más precisas sin haber realizado algo de diseño, sin embargo ésta actividad se realiza en esta etapa porque las situaciones actuales en los medios requieren conocer la estimación lo antes posible, para cualquier toma de decisiones; para disminuir el margen de error en las estimaciones que se realizan en la planeación, se utilizan una serie sucesiva de estimaciones de costos y programación y se redefine cuando se presenta la fase de diseño.

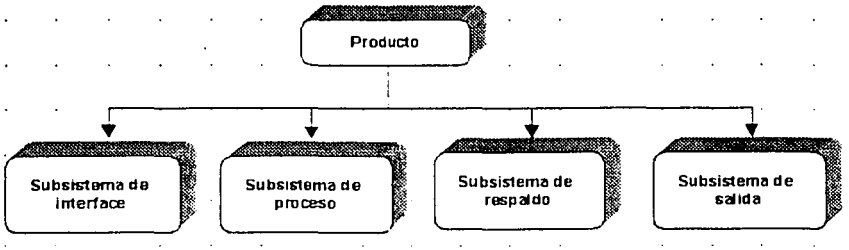
XVII.1.- Estimación a través del Juicio Experto
Una técnica muy utilizada para la estimación de costos es aplicar un juicio del costo basado en la experiencia, en el conocimiento anterior y en la habilidad comercial de una o varias personas de la empresa. La técnica además es del tipo jerárquica hacia abajo, es decir, se enfoca primero en los costos del sistema, de la configuración, del control de la calidad, de la integración, del entrenamiento al personal y de la documentación. Su funcionamiento consiste en realizar estimaciones de costos mediante la comparación del proyecto actual, con respecto a proyectos anteriores similares, cada experto emite su juicio y se discuten las diferencias entre todos los miembros del grupo; esta discusión debe ser lo más imparcial posible con respecto a individuos que dominen o tengan una fuerte influencia hacia el grupo.

XVII.2.- Estimación por medio de la técnica DELFI
Esta técnica es similar a la de Juicio Experto, con la diferencia de que no existe una confrontación directa entre todos los miembros del grupo, en esta técnica se busca un consenso a través de varias rondas de estimación a través de un coordinador y se tiene la ventaja de no contar con los efectos negativos de las reuniones en grupo.

pasos a seguir en la técnica DELFI:
1.- El coordinador proporciona a cada experto la documentación del sistema y una papeleta para que escriba su estimación inicial.
2.- Los expertos escriben y justifican su estimación personal de forma anónima y pueden consultar con su coordinador, pero no entre ellos.
3.- El coordinador realiza un resumen de todas las estimaciones proporcionadas, incluyendo datos o razonamientos incoherentes y extraños por parte de los expertos.
4.- El coordinador prepara un resumen de las estimaciones , sin incluir los razonamiento que se utilizaron para obtenerlas.
5.- El coordinador convoca a una reunión, para analizar las estimaciones en donde se observen mayores diferencias.
6.- Se realiza otra estimación por parte de los expertos de manera anónima, y del proceso 4 al 6 se repite tantas veces como sea necesario, hasta que las estimaciones tengan un comportamiento similar.

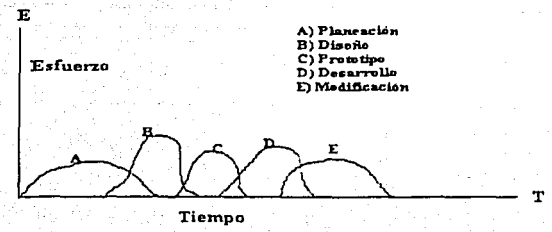
Se puede dar el caso en que nunca se lleguen a obtener estimaciones uniformes, entonces el coordinador deberá analizar los aspectos relacionados con cada experto y determinar así las causas de tales diferencias pudiendo solicitar información adicional.

XVII.3.- Estimación a través de Estructuras de División de Trabajo (WBS)
Para este tipo de estimación se divide en forma de organigrama las diferentes actividades o los procesos principales del sistema de referencia. El organigrama jerárquico identifica claramente todos los componentes del proceso o producto e indica la manera en que estos se encuentran relacionados; los costos se obtienen mediante la asignación del costo de cada componente individual en el organigrama y posteriormente se suman para obtener el costo final.



XVII.4.- Estimación del nivel de contratación (Modelo de estimación de Putnam)
Las personas que intervienen en un proyecto de programación no es constante, pues al inicio, y regularmente en el análisis y la planeación intervienen un pequeño grupo de individuos ; el diseño arquitectónico lo realiza un grupo mayor de personas, y el diseño detallado lo realiza un grupo mas grande de personas. En el mantenimiento del sistema en su etapa inicial pueden intervenir un número considerable de programadores, pero si con el tiempo no existen mejoras o actualizaciones importantes entonces el número de personas designado al mantenimiento se tendrá que ver reducido.

Según P. Nordem los proyectos de investigación y desarrollo siguen un ciclo de vida de esfuerzo y tiempo representado en la siguiente gráfica :

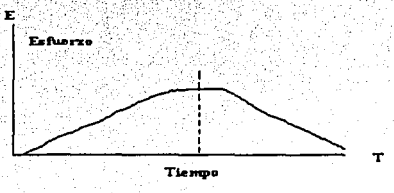


La suma de las áreas bajo estas curvas se aproximan a la ecuación de Rayleigh , en donde cualquier punto de la curva representa el número equivalente de personal de tiempo completo, trabajando en ese momento.

Curva de Rayleigh de esfuerzo contra tiempo

$$E = \frac{K}{td^2} \; t \; e^{\frac{-t^2}{2td^2}}$$

Donde *td* = Es el tiempo en que la curva alcanza su valor máximo
y *K* = Es el área total bajo la curva, que representa el esfuerzo total requerido por el proyecto



Putman investigó que el número de personas requeridas en el ciclo de vida de un proyecto tenía un comportamiento parecido y que se puede obtener una ecuación de la curva Rayleigh-Nordem que relaciona el número de líneas esperadas (L) con el esfuerzo(E) y el tiempo de desarrollo (T).

$$E = \frac{L^3}{C^3 \quad T^4}$$

Donde "C " es una constante del estado de la tecnología, y se puede obtener del entorno propio y a partir de desarrollos anteriores, aunque algunos valores típicos son los siguientes :
C= 2000 se ocupa para entornos pobres de desarrollo, sin metodología, con una documentación y revisiones pobres y modos de ejecución no interactivos.
C= 8000 Para un buen entorno de desarrollo, aplicando una buena metodología, adecuadas documentaciones y revisiones y modos de ejecución interactivos.
C= 11000 para excelentes entornos de programación, con el uso de herramientas y técnicas automáticas.

Entre otras cosas Putnan observó que el área bajo la curva de Rayleigh en cualquier intervalo representa el esfuerzo total utilizado durante ese intervalo y que el 40% se encuentra a la izquierda del área bajo la curva y que el otro 60% a la derecha

XVII.5.- Estimación de los costos de mantenimiento del Software
Gran número del personal directivo piensa que un desarrollo termina cuando el producto es liberado por primera vez, y que los mantenimientos subsecuentes solo llevarán una pequeña parte del tiempo, sin embargo en la realidad esto no es así, pues en los mantenimientos se suele ocupar un 40, un 60 o hasta un 90% del esfuerzo total del ciclo de vida de un proyecto. El mantenimiento comprende básicamente la corrección de problemas en los programas, la inclusión de mejoras a las funciones existentes y la adaptación de nuevos ambientes al proceso. Dentro de las actividades del mantenimiento se puede distribuir el esfuerzo de la siguiente forma:

Actividad	% de esfuerzo
Mejoras	51.3
Mayor eficiencia	4.0
Mejor documentación	5.5
Mejoras para el usuario	41.8
Adaptación	23.6
Datos de entrada y archivos	17.4
Equipo y sistema operativo	6.2
Correcciones	21.7
Arreglos de emergencia	12.4
Arreglos programados	9.3
Otros	3.4

Es de suma importancia que en la fase de planeación del proyecto, se conozca el número de programadores que se requerirán de tiempo completo en el mantenimiento, y esto se puede obtener por medio de el número de líneas de código que se mantendrán, entre el total de líneas de código que puede mantener el programador, es decir, si se necesita desarrollar 120 mil líneas de código (120 KLDC) y cada programador puede mantener hasta 30 KLDC , entonces se ocuparán 4 programadores para el trabajo. El esfuerzo de mantenimiento de la actividad (EACT) puede obtenerse con el número de instrucciones agregadas y modificadas divididas entre el número total de instrucciones:

EACT = (LDCagregadas + LDC modificadas) / LDC totales

El esfuerzo de mantenimiento (EACT) multiplicado por el número de meses de programador empleados durante el periodo específico de desarrollo (MDEV) nos dará como resultado el número de meses requeridos para el mantenimiento (PM_m)

$$PM_m = (EACT) \quad (MDEV)$$

La ecuación anterior puede sufrir una modificación a través de la inclusión de un factor de ajuste, el cual determinará el esfuerzo requerido en el mantenimiento (FEACT) , por lo que una cuidadosa atención y el empleo de técnicas modernas de programación usadas en la etapa de desarrollo puede disminuir la cantidad de esfuerzo utilizado en el mantenimiento.

$$PM_m = (EACT)(MDEV) \quad (FEACT)$$

XVII.6.- Herramientas automáticas de Estimación

Para obtener estimaciones de esfuerzo, costos y tiempo en proyectos también existe software especial que nos puede dar una idea de lo que se tiene que invertir en cada nuevo desarrollo, aunque existen varias herramientas automáticas de estimación casi todas requieren de una o mas de los siguientes datos:

1.- Una estimación cuantitativa del tamaño del proyecto (Líneas de código) o una estimación basada en la funcionalidad del desarrollo (Puntos de función)

2.- Una estimación cualitativa del desarrollo basada en la complejidad del producto, en el nivel de tecnología, en la fiabilidad de los programas, etc.

3.- Características de acuerdo a la capacidad de los programadores y al entorno de desarrollo.

A partir de estos datos el modelo resultante proporciona estimaciones de esfuerzo requerido, de los costos, de la carga de personal y en algunos casos proporciona el calendario mensual de desarrollo, el riesgo asociado, la configuración del hardware y gráficas.

Una herramienta muy completa para la estimación es la SPQR/20 desarrollada por Software Productivity Research , Inc., la cual mantiene un diálogo con el planificador obteniendo la información apropiada sobre el proyecto; algunas respuestas que tiene que especificar el planificador son acerca de:

- El tipo de proyecto (Programa nuevo / Mantenimiento)
- El ámbito del proyecto (Prototipos, módulos reutilizables)
- Los objetivos (Incluyendo tiempos y calidad)
- La clase de proyecto (Programa personal o institucional)
- Tipo de aplicación (Interactiva, sistema experto)

- Entorno de trabajo
- Requisitos del programa
- Requisitos de diseño
- Tiempo de respuesta
- Experiencia del personal
- Porcentaje de código fuente reutilizable
- Lenguaje de programación
- Nivel de complejidad en los algoritmos, el código y los datos

Las técnicas de estimación empíricas y automáticas pueden diferir con los resultados reales, por lo que dichas técnicas solo deben ser un punto de partida para la estimación del proyecto, es decir, la estimación de un proyecto de software nunca será una ciencia exacta, por lo que para obtener estimaciones más acertadas, generalmente se deben de promediar los resultados de dos o tres técnicas de estimación .

XVIII.- ANÁLISIS E IDENTIFICACIÓN DEL RIESGO

Una vez realizada la estimación de nuestro proyecto, se debe tomar en cuenta el riesgo al que debemos enfrentarnos durante y después del proyecto, ya que el riesgo es uno de los factores inevitables de todo proyecto

Para identificar el riesgo de un proyecto se deben definir concretamente todas aquellas actividades que pueden ocasionar retrasos o pérdida de calidad en los proyectos.

Los riegos también pueden dividirse en tipos de riesgos: Riesgos del proyecto, riesgos técnicos y riesgos del negocio. Los riesgos del proyectos abarcan problemas de presupuesto, de calendarización, de organización de recursos, problemas con el cliente, problemas con los requisitos e impactos. Los riesgos técnicos aparecen cuando el problema es mas complejo de resolverse que lo esperado, y estos pueden ser de interface, de diseño, de verificación y de mantenimiento. Los riesgos de negocio son enfocados a problemas de mercadotecnia, a estrategias o políticas inacordes con el producto de software y hasta pérdidas presupuestarias.

Por ejemplo, para identificar si estamos trabajando con la plataforma, el software y la calidad adecuada, nos podemos ocupar en responder algunas de las siguientes preguntas.

Control:
- Se llevará el control de un inventario de los programas a desarrollar ?
- El inventario de sistemas identifica todos los programas de cada sistema ?
- Que tan actualizadas se encuentran las herramientas de desarrollo ?

Seguridad (Ambiente):
- Como se va a dotar de seguridad al ambiente desarrollador ?
- Quien asignará y controlará los passwords
- Quién dará atención en caso de necesitar las claves de seguridad y de no contar momentáneamente con la gente que controla la seguridad de sus ambientes ?
- Como estarán protegidos los programas fuente y ejecutables.
- Como se respaldarán y guardarán los fuentes y ejecutables ?
- Como se llevará el control de las versiones de los módulos o sistemas ?

Seguridad (Información):
- Como se mantendrá la seguridad de la información ?
- Cuantos niveles de seguridad de la información protegen sus archivos ?
- Como se organizarán y mantendrán los respaldos fuera de sitio ?
- Como se protegerán los archivos contra manipulaciones ?

Seguridad (Herramientas):
- Como se dotarán de seguridad a los equipos de desarrollo y de proceso ?
- Como se respaldarán los principales equipos en que almacenan la información ?
- Los equipos y lenguajes cumplirán con las expectativas de la empresa ?
- Las regulaciones oficiales estarán cabalmente cubiertas ?
- Como se asegurará el no perder información de archivos o sistemas en caso de adquirir virus en los equipos ?
- Se tiene un sistema de respaldo de la información ?

Seguridad (Desarrolladores):
- Que estrategias se seguirán para propiciar la permanencia de los desarrolladores especialistas ?
- Se han estructurado reemplazos si súbitamente no se cuenta con algún desarrollador ?
- Como se asegurará la actualización técnica de los desarrolladores ?

Calidad
. Como se garantizarán los niveles de calidad de los desarrollos particulares ?
- Que herramientas se utilizarán para medir y elegir la alternativa adecuada en sus desarrollos particulares ?
- Se esta trabajando con las herramientas institucionales de la Empresa ?
- Se recibe soporte técnico de parte del proveedor para el manejo del equipo ?
- Se cuentan con las licencias adecuadas para la manipulación del Software ?
- Se recibe la correcta capacitación del lenguaje a utilizar ?
- Que tan actualizados tienen sus herramientas de desarrollo ?

Preguntas de seguridad para las instalaciones:
- Existen fallas eléctricas o caídas y subidas de voltaje ?
- Los equipos se encuentran en una zona de alta sismicidad ?
- El ambiente de trabajo es el adecuado para el equipo (Temperatura, humedad) ?

El impacto y la probabilidad de ocurrencia del riesgo son dos factores determinantes en la proyección del riesgo, pues un factor de riesgo con un peso muy alto de impacto pero una probabilidad muy baja de que ocurra no debería absorber una cantidad significativa de tiempo de gestión, por otro lado, los riesgos de alto impacto con probabilidad de ocurrencia media o alta se deben "evaluar" y definir los niveles de referencia de ese riesgo, para la mayoría de los proyectos el costo, la agenda y el rendimiento representan 3 puntos de referencia de riesgo. Si alguno de estos puntos excede de manera negativa o la combinación de algunos de ellos crea problemas, entonces se debe de analizar el punto de ruptura o de referencia, en donde continuar o no con el proyecto son decisiones igualmente aceptadas.

La fase de planeación de un proyecto no termina con lo que hemos visto, si no que aquí se empiezan a sentar las bases de lo que es el desarrollo de un proyecto, para que posteriormente se continúe la planeación del diseño, el desarrollo y el mantenimiento, los cuales se muestran a continuación.

XIX.- PLANEACIÓN EN EL DISEÑO DE SOFTWARE

El diseño es la parte central de la Ingeniería de Software, pues representa el enlace entre los requisitos de la programación y una codificación que satisfaga con dichos requisitos, durante el diseño se desarrollan, se revisan y se documentan los progresivos refinamientos de las estructuras de datos, de la estructura del programa y de los detalles de cada procedimiento. Técnicamente el diseño se divide en diseño externo (De datos) , diseño arquitectónico y diseño detallado y desde el punto de vista administrativo el diseño se divide en preliminar y detallado (ó externo e interno)

El diseño externo requiere de concebir, planear y especificar las características de un producto y estas características incluyen la definición de despliegues en pantalla, la forma de los reportes, la identificación de entradas y salidas de datos, así como satisfacer los requerimientos de desempeño y determinar las características funcionales y estructurales del producto. El diseño interno incluye una especificación de la estructura arquitectónica, los detalles de los algoritmos, las estructuras de datos y los planes de prueba.

El diseño arquitectónico se encarga del establecimiento de las relaciones e interconexiones entre las funciones, los datos y el almacenamiento de ellos, es decir, define las relaciones entre los principales elementos estructurales del programa, además se ocupa del refinamiento del sistema,

de la identificación de funciones internas del proceso, de la descomposición de funciones y de la definición de cadenas de datos y almacenamientos. El diseño detallado se encarga básicamente de las representaciones algoritmicas del software y del refinamiento de la representación arquitectónica;

Un buen diseño es la plataforma que sostiene a los siguientes niveles de desarrollo del proyecto :

XIX.1.- Conceptos de diseño

Los conceptos de diseño de software proporcionan la base necesaria para que éste se lleve de la mejor forma e incluyen la abstracción, ocultamiento de información, la estructura, guardado de información, modularidad, concurrencia, verificación y los aspectos estéticos en el diseño.

XIX.1.1.- Abstracción

La Abstracción es una habilidad que puede utilizar el ingeniero de Software durante el diseño para representar y confrontar los conceptos a los cuales se enfrenta, es aquí en donde reduce cualquier complejidad y se permite organizar y dirigir los procesos del pensamiento al posponer las condiciones estructurales y algoritmicas hasta que las características funcionales, las cadenas de datos y los almacenamientos hayan quedado definidos. Existen varios niveles de abstracción en donde el nivel superior muestra una solución en términos amplios y utilizando el lenguaje del entorno del problema, mientras que el nivel de abstracción más bajo es el que da la solución al problema desde el código fuente.

Tres son los tipos de abstracción que permiten al ingeniero controlar la complejidad del proceso del diseño, estos son: las abstracción funcional, la abstracción en los datos y la abstracción en el control, la primera es un mecanismo en donde se parametrizan y enlazan programas y subprogramas, por lo que las rutinas visibles se comunican con otros grupos y las rutinas escondidas sirven de apoyo hacia las rutinas principales visibles. La abstracción en los datos incluye las especificaciones en los tipos de datos o de los objetos por medio de especificar las operaciones permitidas sobre ellos, eliminando el manejo y los detalles de su representación. La abstracción del control es utilizada para poder llevar a cabo el flujo del control mediante proposiciones de condición y decisión (SI ENTONCES, HAZ MIENTRAS; ETC.) por lo que permiten la especificación de programas secuenciales, de manejadores de excepciones, de subrutinas y de programas concurrentes sin preocuparse en los detalles exactos de la codificación.

XIX.1.2.- Ocultamiento de Información

Este enfoque se basa en que cada módulo del sistema debe ocultar los detalles internos de sus procesos, comunicándose los módulos mediante interfaces definidas, es decir, los módulos deben especificarse y diseñarse de tal forma que la información (procedimientos y datos) contenida en ellos no se pueda ver en otros módulos independientes. Un ocultamiento verdaderamente efectivo es aquel en donde existen módulos independientes y solo se comunican mediante la información necesaria para realizar la función de software correcta.

XIX.1.3.- Estructura

La utilización de una estructura nos permite ver a un sistema grande en términos de unidades mas pequeñas y manejables y con una clara definición de las relaciones entre las diferentes partes del sistema. En la estructura se define la jerarquía de control e incluyen referencias hacia todos los submódulos, de tal forma que la representación se puede ver por capas

XIX.1.4.- Modularidad

La modularidad consiste básicamente en dividir el software en tareas específicas llamadas módulos (funciones, subrutinas, procedimientos) , que a su vez también pueden ser subdivididas; La modularidad mejora la claridad en el diseño, facilita la codificación, las pruebas, el mantenimiento, la documentación y permite el desarrollo paralelo de las diferentes partes del sistema. La modularización consiste en descomponer a un sistema en unidades funcionales e implantar un orden jerárquico en el uso de sus funciones. Los sistemas modulares contienen unidades e interfaces claramente bien definidas y manejables por lo que en un sistema no modular podría ser imposible su correcta comprensión. Un módulo de programación tiene generalmente las siguientes características :

A).- Contienen instrucciones, estructuras de datos y lógica de procesamiento.

B).- Cada módulo puede ser compilado y almacenado por aparte.

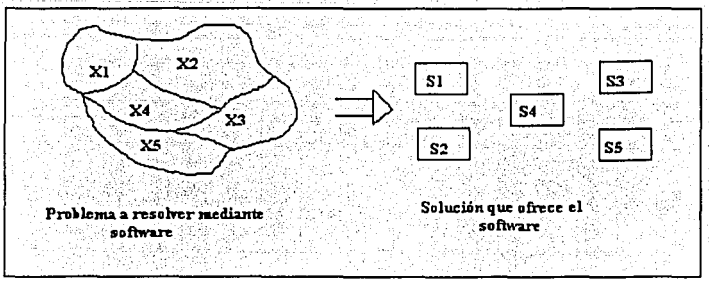
C).- Los módulos pueden ser incluidos dentro de los programas y éstos a su vez pueden usar a otros módulos

D).- Los módulos pueden ser invocados mediante una llamada que contenga ciertos parámetros

Mediante la modularidad se aplica el lema "Divide y vencerás" y es más fácil resolver un problema complejo al descomponerlo en varias partes, sin embargo conforme crece el número de módulos o subrutinas, el esfuerzo asociado al manejo de interfaces entre módulos también crece, por lo que se debe buscar un punto de equilibrio entre una modularización excesiva y una pobre modularización.

XIX.2.- Arquitectura de software

La arquitectura del software se forma de la relación entre un problema real y la solución que se le dé mediante componentes de software, por lo que representa la transición entre el análisis de requisitos y el diseño



La arquitectura de software se compone de dos estructuras básicas : La estructura del programa y la estructura de datos, la primera representa la jerarquía de los módulos e indica los niveles en los que se controlan los distintos bloques o módulos, "no" representa aspectos detallados del software como la ocurrencia u orden de instrucciones, la secuencia de procedimientos, ni la repetición de operaciones.
El diseño arquitectónico tiene como objetivo el desarrollar una estructura modular de programas y representar las relaciones de control entre esos módulos, además mezcla la estructura de programas y la estructura de datos y define las interfaces que facilitan el flujo de los datos a lo largo del programa.

XIX.3.- Estructura y diseño de datos
Una actividad importante en el diseño es la de identificar los módulos de programa que deben operar directamente sobre las estructuras de datos lógicas, por lo que independientemente de la metodología de diseño utilizada, los datos bien diseñados conducen a una mejor estructura de programa, a una modularidad efectiva y por consecuencia a una reducida complejidad procedimental. La estructura de datos a utilizar la determina el diseñador y esta representa la relación directa existente entre la información del mundo real y los elementos individuales de datos; la estructura de datos formula el tipo de organización, los métodos de acceso, el grado de asociatividad y las alternativas de almacenamiento para la información. Las estructuras de datos clásicas son : El elemento escalar, el arreglo secuencial, la lista enlazada, el arreglo n-dimensional y el árbol jerárquico.

Los principios para especificar datos son:
- Deben especificarse todas las estructuras de datos y las operaciones que se han de realizar sobre cada una de ellas
- Se debe establecer y utilizar un diccionario de datos para definir el diseño de los datos y del programa

- Se deben posponer las decisiones de diseño de datos de bajo nivel hasta más adelante en el proceso de diseño.
- La representación de una estructura de datos solo debe ser conocida por los módulos que hagan un uso directo de los datos contenidos en la estructura
- Se debe desarrollar una biblioteca de estructuras de datos útiles y de las operaciones que se les pueden aplicar.
- El diseño de software y el lenguaje de programación a utilizar deberán soportar las estructuras de datos utilizadas

XIX.4.- Independencia Funcional en el diseño
La independencia funcional trata de diseñar módulos maximizando la utilización de funciones específicas y claras (**Cohesión**) e intenta minimizar la iteración excesiva entre la comunicación de los módulos (**Acoplamiento**) , es decir, trata de diseñar software de tal forma que cada módulo se ubique en realizar una subfunción específica de los requisitos y maneje interfaces sencillas. La independencia funcional es una de las claves del diseño para desarrollar software con calidad, ya que los módulos independientes son más fáciles de desarrollar, probar y mantener, debido a que se centran solo en los efectos secundarios producidos en el código y se pueden reutilizar en cualquier momento.

XIX.5.- Diseño procedimental o detallado
El diseño detallado se puede empezar, una vez que hemos terminado de definir la estructura de datos y la estructura del programa, En el diseño procedimental se necesita detallar más a fondo cada procedimiento que pertenece al proyecto y para ello se utilizan distintas técnicas específicas que representan este tipo de diseño:

XIX.6.- Notaciones de diseño
El diseño se apoya de esquemas de representación y de notaciones gráficas para representar una solución particular de cada módulo. Ambas herramientas son de fundamental importancia pues nos ayudan a aclarar las características externas de un sistema de programación , las especificaciones del diseño arquitectónico que describen la estructura del sistema y las especificaciones del diseño detallado, que describen el flujo y la representación algorítmica de cada módulo del sistema: Las notaciones más utilizadas son: Diagramas de Flujo, Diagrama de cajas, Diagramas HIPO y el Seudocódigo.

Cualquier notación para el diseño puede ser valiosa en el proceso de diseño, sin embargo también suele ser complicada si ésta fue aplicada de manera incorrecta; una buena notación de diseño debe conducir a una representación de los procedimientos , para que éstos sean fáciles de comprender y de revisar, además debe facilitar la tarea de codificación de forma que el código se obtenga como un producto natural del diseño.

XIX.6.1- Diagramas de Flujo y formas de representación
Los diagramas de flujo representan la notación gráfica más comúnmente utilizada, pues en ella podemos mostrar cualquier flujo de datos y procesos, representándolos al detalle de la toma de decisiones. Un diagrama de flujo muestra el camino que siguen los datos a través de bloques (Procesos), en donde éstos pueden ser rutinas secuenciales bien definidas ó macro-bloques que nos indican en forma genérica las acciones y decisiones globales.

Para representar un paso de procesamiento se utiliza una caja en forma de rectángulo, una toma de decisión se especifica mediante un rombo (Si-Entonces-Si no), la forma de repetición "Hacer Mientras" pregunta por una condición, y se ejecuta repetidas veces mientras la condición se cumpla, otra variante de ciclo de repetición es la forma "repetir-hasta" en donde primero un bloque es ejecutado, y al final se pregunta por una condición, si la condición se cumple se volverá a ejecutar dicho bloque. Estas construcciones estructuradas pueden anidarse unas entre otras y poder formar así un diagrama más completo, sin embargo existirán procesos en los cuales se requiere salir del conjunto de estructuras anidadas, lo cual se resuelve con la instrucción de escape "Exit",

aunque se corre el riesgo de contrarrestar la facilidad de lectura e incrementar los errores y mantenimientos; Para contrarrestar éstas desventajas podemos rediseñar nuestro diagrama de tal forma que no sea necesaria la salida de escape, o en última instancia se puede utilizar el método de escape pero de una forma controlada. La representación de estos diagramas la podemos observar en el capítulo uno.

XIX.6.2.- Diagramas de Cajas

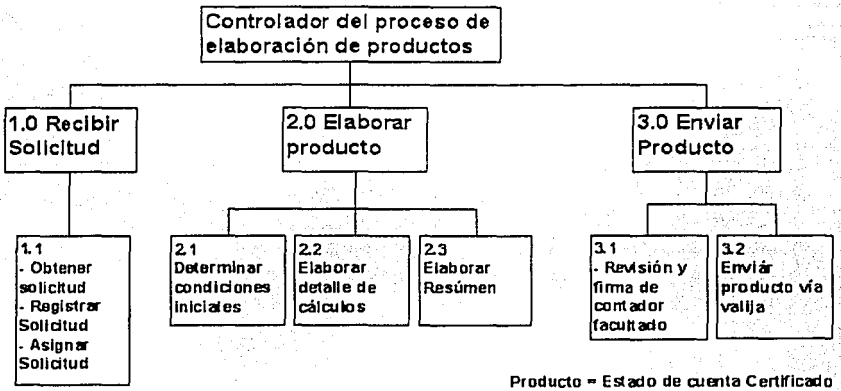
Los diagramas de cajas fueron desarrollados originalmente por Nassi y Schneiderman y mejorados después por Chapin; básicamente fueron implementados para no violar las construcciones estructuradas, por lo que no se permiten las bifurcaciones de escape "EXIT" , en ninguna caja utilizada. Este tipo de representación tiene las siguientes características:

1).- Las estructuras de control están claramente bien definidas
2).- No se permite la transferencia arbitraria del control entre las estructuras
3).- Se pueden identificar fácilmente las variables globales y locales
4).- La representación de la recursividad no presenta mayor problema

XIX.6.3.- Diagramas HIPO (Hierarchy - Input - Proces - Output)

Este tipo de diagramas son utilizados como medios de representación del diseño "jerárquico" de arriba hacia abajo, y también sirven como auxiliar en la documentación de productos para los clientes. Este tipo de representación está compuesto por una tabla visual del contenido, un conjunto de diagramas generales y un conjunto de diagramas a detalle. El conjunto de diagramas generales especifica los procesos funcionales, describiendo básicamente las entradas, los procesos y las salidas de cada parte funcional; también un diagrama general puede identificar a los diagramas de detalle subordinados, los cuales contienen las funciones específicas de cada proceso.

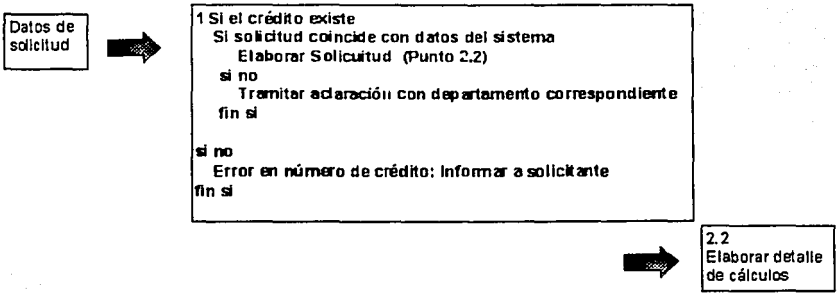
DIAGRAMAS GENERALES TIPO HIPO



Producto = Estado de cuenta Certificado de un crédito hipotecario

DIAGRAMAS A DETALLE TIPO HIPO

Diagrama 2.1 Determinar condiciones iniciales



XIX.6.4.- Seudocódigo

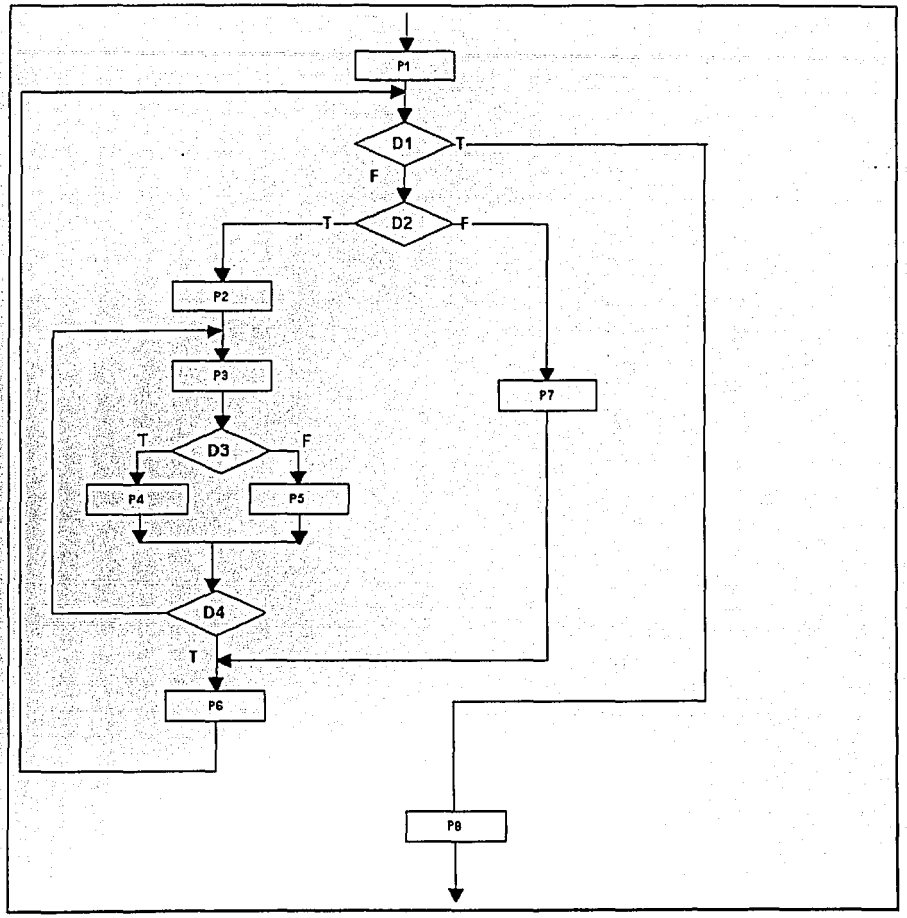
El Seudocódigo puede ser utilizado en cualquier nivel de abstracción y éste se representa básicamente con el uso de un lenguaje natural (Inglés o español) y algunas funciones y estructuras de control utilizados en los lenguajes de alto nivel, con esto y con la utilización de las sangrías, se puede decir que es lo más parecido a la codificación. Para tener un mejor control sobre el seudocódigo a veces es necesario tener ciertas reglas a seguir:

- Tener una sintaxis fija de palabras clave que nos permitan construir todas las estructuras de control, declaración de datos y establecimiento de llamadas a subprocedimientos
- Tener una sintaxis libre en el lenguaje natural elegido, para expresar las características del procesamiento con claridad de entendimiento.
- Considerar las facilidades de definición de datos escalares, arreglos, listas enlazadas y árboles.

Diagrama de flujo estructurado y su equivalente en seudocódigo:

Donde:
P = Proceso
D = Desición

```
P1
Mientras D1 repetir
  Si D2 Entonces
    s2
    repetir
      P3
      Si D3 entonces
        P4
      Si no
        P5
      Fin Si
    Hasta D4
  Si no
    P7
  Fin si
  P6
Fin Mientras
P8
```



XIX.7.- Metodología de diseño

Como se había comentado, el diseño es el punto de enlace entre el análisis de requisitos y el desarrollo de la solución, y esto comprende el desarrollo de una visión del sistema, el establecimiento de una estructura, la definición de las cadenas de datos, la descomposición del sistemas en módulos, el establecimiento de las relaciones e interconexiones entre componentes, así como la especificación de los detalles a través de algoritmos. Todo lo anterior se puede hacer utilizando algunas técnicas o metodologías de diseño, que más bien son herramientas que se pueden o no ajustar a nuestras necesidades de diseño y también haciendo uso de nuestra creatividad como marco de apoyo y referencia.

Las técnicas de diseño están basadas en las jerarquías de "hacia abajo" y "hacia arriba" ; mediante el primer enfoque se concentran los esfuerzos en las actividades principales, y poco a poco se va descomponiendo cada parte, para después poner mayor consideración en los detalles más específicos , mientras que al diseñar hacia arriba se deben identificar al conjunto primitivo de objetos, acciones y relaciones que proporcionarán una base para la solución del problema ; en esta estrategia se combina el uso del lenguaje de programación (funciones), estructuras de datos e interconexiones que darán soluciones "hacia atrás" a nuestro sistema. En la práctica existen soluciones que no necesariamente deben utilizar solo un tipo de estrategia, si no más bien ambas técnicas se deben de ocupar para obtener la mejor solución. Existen específicamente metodologías que conllevan este tipo de técnicas complementadas con otro tipo de características, y éstas pueden ser: el refinamiento por pasos, los niveles de abstracción, el diseño estructurado, etc.

El siguiente paso para llevar a cabo una planeación con calidad es la codificación estrictamente planeada, la cual veremos a continuación.

XX.- PLANEACIÓN DE LA CODIFICACIÓN

Como hemos visto, las necesidades y restricciones de los clientes se registran en la especificación de requisitos; los requisitos proporcionan el marco de trabajo para el diseño estructural y el diseño detallado se forma a partir del diseño estructural. Ahora entramos a la etapa de la elaboración del código, el cual se desarrolla a partir de los diseños estructural y detallado. Posteriormente se elaboran los planes de prueba, manuales de usuario, programas de entrenamiento, instrucciones de instalación y procedimientos de mantenimiento los cuales evolucionan a través de todo el ciclo de desarrollo.

La codificación es el proceso de transcribir al diseño detallado en algún lenguaje de programación. Cuando el diseño ha seguido correctamente los pasos de la ingeniería de software, entonces la codificación se debe de dar como un proceso simple y natural. El objetivo principal de la codificación aparte de resolver el problema, es el de ser claro y comprensible, pues la legibilidad del código fuente facilita factores de depuración, pruebas y mantenimientos, los cuales consumen gran parte del tiempo y presupuesto en las etapas finales del proyecto.

La correcta interpretación en el diseño detallado para traducir el problema a un lenguaje de programación es esencial, pues de ello se deriva el código del programa que dará la solución que deseamos.

Otro factor importante en el proceso de codificación es el lenguaje de programación utilizado, pues tiene un impacto directo en la legibilidad, la calidad y la eficiencia de nuestro programa ; Un lenguaje de programación complejo y con restricciones puede producir un código fuente obscuro que resulte difícil de probar y mantener, por lo que lo ideal es elegir un lenguaje que se ajuste a las necesidades propias del sistema a desarrollar y no se tenga un único lenguaje para programar en donde los ingenieros de desarrollo
se tengan que ajustar a las limitaciones impuestas por dicho lenguaje.

Independientemente de las buenas características que ya nos presentan los lenguajes de programación modernos, para alcanzar la legibilidad, sencillez y elegancia en el proceso de codificación, tenemos que utilizar un estilo de codificación apropiado, practicar técnicas de codificación estructurada, utilizar documentaciones internas, así como el uso de estándares de codificación y documentos de apoyo.

XX.1.- Estilo de codificación
El objetivo principal de utilizar un estilo de codificación apropiado es el de hacer entendible al código fuente y la consulta mínima del diseño detallado para poder entender lo que hace un programa. Un buen estilo de programación puede superar las deficiencias de un lenguaje de programación primitivo, mientras que un estilo inadecuado puede ocasionar grandes confusiones aunque se esté trabajando con un excelente lenguaje.

XX.1.1.- GO TO disciplinado
La instrucción GO TO se utiliza para lograr las salidas múltiples y prematuras de alguna región de código y para transferir el control a las rutinas manejadoras de los errores, sin embargo para garantizar su sencillez es recomendable utilizarlo solo en trasnferencias de control hacia adelante y dentro de una región local de código, es decir, no lo debemos utilizar para transferir el control hacia el inicio o a regiones externas del programa.

XX.1.2.- Definición de variables y declaración de datos en la codificación
Cuando se realiza un mantenimiento es muy importante que se entienda a simple vista lo que el código hace, y para lograr esto podemos empezar a definir a las variables que manejaremos en nuestro código. Las variables deben ser lo mas parecidas al contexto que se trate, por ejemplo :

```
I = ( C * I * D) /360

Int_Vdo =  ( Cap_Vdo * Tas_Mora * Días ) / 360

Interés_Vencido = (Capital_Vencido * Tasa_Moratoria * Días ) / 360
```

Las tres sentencias anteriores representan lo mismo y pueden ser muy claras para el programador que las escribió, sin embargo la 1a. sentencia a simple vista puede resultar mal interpretada por otro programador, o incluso por el mismo que desarrollo el programa durante algún mantenimiento posterior. Es decir debemos utilizar las variables más parecidas al contexto que trate siempre y cuando el lenguaje de programación lo permita

La complejidad y la organización de las estructuras de datos se definen durante el diseño, pero el estilo de la declaración de datos se determina cuando se genera el código, por lo que en esta etapa es conveniente estandarizar esta declaración, para un mayor entendimiento:
Declarar e inicializar todas las variables (y es recomendable utilizar comentarios para conocer el significado de cada variable que resulte compleja), declarar todos los bloques de datos globales, declarar todos los arreglos públicos y sus dimensiones e incluir todas las declaraciones de los archivos a utilizar . Cuando se declaran múltiples variables dentro de una misma sentencia es conveniente acomodarlos en orden alfabético ó por orden de aparición durante el programa.

XX.1.3.- Claridad y eficiencia en sentencias y estructuras de control
Cada sentencia debe ser sencilla y directa, siempre se debe de buscar un equilibrio entre la legibilidad y la eficiencia, aunque un principio fundamental nos dice que no hay que sacrificar la claridad ni la legibilidad en aras de una eficiencia que no sea esencial (Muchos lenguajes de programación permiten múltiples sentencias en una misma línea, pero el

ahorro en espacio que esto implica esta difícilmente justificado por la pobre legibilidad del resultado

Recomendaciones para incrementar la claridad en las estructuras de control es recomendable :

- Eliminar el uso de comparaciones negativas
- Evitar el anidamiento excesivo de bucles
- Evitar el uso de complicadas comparaciones condicionales
- Utilizar el uso de los paréntesis para un mejor entendimiento de las condiciones
- Usar espacios y/o símbolos claros para incrementar la legibilidad en el contenido de la sentencia

XX.1.4.- Documentando el código

La forma de comunicación entre el programador y otros lectores del código fuente son los comentarios internos, los cuales son un medio importante de entendimiento del código fuente; los lenguajes actuales de propósito general tienen la posibilidad de expresar las ideas generales y particulares de lo que realiza el código, mediante la inclusión de sentencias con el lenguaje natural, por lo que pueden resultar una clara guía de comprensión hasta para el mismo desarrollador durante la fase de codificación y mantenimiento.

Existen 2 modos de comentar los programas, los comentarios de prólogo y los comentarios descriptivos, los primeros deberán de ir al principio de cada módulo y nos describirán de manera general los objetivos, las entradas y salidas de nuestro código, descripciones de las interfaces a utilizar, y la historia del desarrollo del módulo.

A diferencia de los comentarios de prólogo, los **comentarios descriptivos** se ubican a lo largo de todo el programa y éstos deben de describir bloques de código y no transcribir cada línea del programa, por lo que deben de proporcionar un valor agregado al programa, de otro modo solo estarían parafraseando nuestro código sin ningún sentido (Excepto cuando existan sentencias difíciles de comprender se podrán transcribir línea a línea las instrucciones al lenguaje natural). Los comentarios descriptivos deben de distinguirse del código normal, utilizando líneas en blanco, tabulaciones, identificadores, colores diferentes o alguna otra característica que nos otorgue el lenguaje de programación en uso, también deben de ser correctos, ya que resulta peor poner un comentario incorrecto que el no ponerlo, también es recomendable no manejar comentarios largos y confusos para aclarar un código oscuro y complejo, por lo que será preferible reescribir y mejorar el código.

Ejemplo de comentario de prólogo:

```
* Nombre del programa : P_Edita.prg
* Descripción: Edita los tipos de crédito para modificación y/o eliminación
* Ejemplo de llamada:  do P_Edita(d_FecIni, d_FecFin)
* Este programa es llamado por la forma: Catalogo.scx
* Entradas:                    d_FecIni = Fecha Inicial
                               d_FecFin = Fecha Final

* Datos Extras:               Los parámetros son las fechas para la generación del estado de cuenta

* Submódulos utilizados:      Util.prg

*Autor:                       Jorge Ibañez Gutiérrez

*Auditor:                     Alfonso Martinez Ordoñez

*Fecha                        01/JUN/2000

* Modificación                Fecha                Descripción
                               13/JUN/2000          Incremento de la póliza  del 12.30% a partir del 2001
```

En el siguiente ejemplo se utilizan "&&" al inicio de cada línea de comentario para saber que se trata de un comentario descriptivo importante, y también se utiliza el * como línea de comentario para saber que hubo un cambio de instrucción que se quiere mantener como referencia.

```
Ejemplo de comentarios descriptivos
&& Fórmula para calcular el interés moratorio en base a días naturales
f_IntMora = (f_CapVenc * Tasa * 30 / 360)
* f_IntMora = (f_CapVenc * Tasa * i_NumDias /360)
```

XX.2.- Controles del código: Carpeta de desarrollo y unidades de programa.
Una unidad de programa es un módulo o submódulo de código fuente, el cual es lo suficientemente modular para ser probada de forma independiente. La carpeta de desarrollo ó "Notas de Programa" representan documentos que le sirven al desarrollador para organizar sus actividades de trabajo y para conservar toda la información documental de sus unidades de programa, por lo que es responsabilidad de cada programador el documentar y conservar dicho control a través de la carpeta.

La carpeta de desarrollo consta de una portada y varias secciones. la portada es la tabla de contenidos y la hoja de avisos de terminación de los diversos logros asociados con la unidad de programa. La carpeta de desarrollo permanece con la unidad de programa durante todo el tiempo de vida, y se transfiere el control de programador a programador a medida que se cambian las responsabilidades

Finalmente veremos la etapa de mantenimiento del software la cual implícitamente se planifica desde el primer punto de este capítulo, pues a medida que el proyecto se planea adecuadamente desde el principio, entonces el mantenimiento a realizar se ocupará solo para nuevas mejoras.

XXI.- PLANEACIÓN AL MANTENIMIENTO

Cuando se acepta y se entrega un desarrollo de software se piensa que hasta ahí se trabajará con el producto entregado, sin embargo, continúa una actividad encargada de mejorar, adaptar y corregir al sistema, es decir, el sistema se desarrolla y posteriormente pasa a una evolución continua incrementando la numeración de su versión original; Cualquier proceso que evoluciona tiende a dirigirse a dos caminos, o se vuelve obsoleto por no recibir el mantenimiento oportuno ó sufre modificaciones durante casi toda su vida. Cuando se descubre un error o se desea realizar una mejora en el sistema algunas veces ésta se realiza de manera inmediata (dependiendo de la urgencia), o se realizan las modificaciones en base a un calendario de actividades paralelas con otros proyectos, ó simplemente no se hace nada (debido a la falta de prioridad del proyecto o a la falta de recursos), por lo que un fin que se desea alcanzar en el proceso es el de mejorar la facilidad con la que se puedan adaptar los cambios y reducir la cantidad de esfuerzo empleado en el mantenimiento.

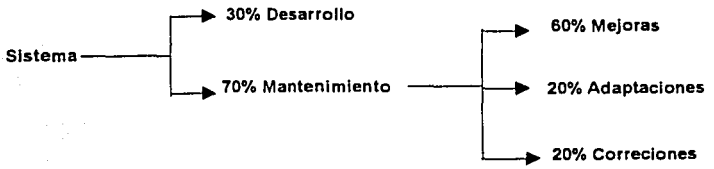
Algunas veces el error es tan grave, que se tienen que abandonar las actividades actuales para corregir de inmediato el problema, sin la evaluación de los posibles efectos secundarios y sin la adecuada actualización en la documentación, sin embargo estas actividades no se eliminan, sino que se llevarán acabo posteriormente, hasta que se haya corregido el problema. Una vez que se haya resuelto la crisis se debe analizar si los cambios realizados no afectan a otros procedimientos del programa, lo cual pudiera generar incluso errores mas serios. Este modo de actuación

inmediata debe representar un porcentaje mínimo en la vida diaria y queda reservado exclusivamente para situaciones de verdaderas crisis.

Existen cuatro tipos de mantenimientos, el primero es el ***mantenimiento correctivo*** el cual se pone en marcha cuando se descubren errores y el sistema aún no se ha puesto en marcha, el ***mantenimiento adaptativo*** surge cuando el entorno externo exige cambios de adaptación (Aparición de nuevos sistemas operativos, nuevas versiones del lenguaje de programación, mejora en periféricos, etc.); el tercer tipo de mantenimiento es el ***mantenimiento perfectivo***, el cual se produce cuando un sistema es utilizado con éxito y después se solicitan nuevas implementaciones y mejoras por parte de los usuarios, quienes retroalimentan al personal de informática, para la elaboración de dichos cambios, el último tipo de mantenimiento y que desgraciadamente casi no se practica es el ***mantenimiento preventivo***, el cual consiste en que antes de que se origine algún cambio, se analice y documente el proyecto de manera general y se tenga de esta forma un mejor control para las nuevas modificaciones, por lo que si en una empresa se desea realizar un mantenimiento preventivo, se deben elegir a aquellos programas que sean susceptibles de cambiar en el futuro cercano y prepararlos para dicho cambio.

XXI.1.- Tiempo invertido en el mantenimiento.
Muchas veces la calidad del mantenimiento y la cantidad de esfuerzo dedicado depende de si existe una documentación completa del proyecto o si se aplicó el proceso de la ingeniería de software en el desarrollo, pues aveces, si solo se dispone de código fuente la actividad de mantenimiento comienza con un pesado análisis del código y quizá son mal interpretadas o difíciles de descubrir las estructuras de datos globales, las interfaces del sistema, el rendimiento del sistema, la estructura del programa, etc. , También es imposible realizar las pruebas de regresión y en ellas observar si las modificaciones afectaron a otros procesos, pues no se cuenta con los registros de las pruebas anteriores. En cambio si en el proyecto a otorgarle mantenimiento se llevó una metodología de software bien definida, entonces se comenzará con una evaluación de la documentación del diseño, se determinan las características estructurales de rendimiento y de interfaz, se desarrolla nuevo código fuente y se realizan pruebas de regresión mediante los registros contenidos en las especificaciones de pruebas, por lo que se puede decir que se mejora la calidad general del cambio y se reduce la cantidad de esfuerzo requerido.

Tiempo promedio invertido en algunos proyectos



XXI.2.- Costos del mantenimiento

Algunas veces los costos de mantenimiento representan aproximadamente el 60% de los gastos de todo el proyecto, y también se presentan costos intangibles de oportunidad, los cuales se presentan cuando se pospone o suspende la oportunidad de un nuevo desarrollo debido a que los recursos se encuentran asignados al mantenimiento de un proyecto, es decir la productividad se ve seriamente afectada de modo que ni los ingenieros desarrolladores, ni los usuarios finales se encuentran 100% satisfechos.

Otros costos intangibles son los siguientes:

- Existen clientes descontentos cuando realizan una petición de modificación al sistema, y ésta no se realiza en el tiempo esperado.
- Existe una disminución en la calidad global del desarrollo debido a los nuevos errores que introducen los cambios por mantenimiento
- Inconformidades en otros esfuerzos de desarrollo al tener que poner a trabajar a la plantilla en mantenimiento de otros proyectos.

XXI.3.- Planeación de las actividades de mantenimiento

Muchas veces no se considera la planeación de las actividades de mantenimiento y no se tiene la visión de que éstas empiezan desde el análisis, diseño y desarrollo del sistema, pues en el análisis se señalan estándares y principios generales para el proyecto de modo que se garantiza la uniformidad de los productos; durante el diseño estructural se debe recalcar la necesidad de realizar procesos claros, modulares y fáciles de mantener y "no" introducir tanto a los criterios de eficiencia y minimización en el espacio de memoria (en los casos que no sea necesario); En la codificación se deben de utilizar construcciones de una sola entrada y una sola salida, se debe observar la identación estándar en las construcciones, se debe adoptar un estilo de codificación simple y se deben contemplar los comentarios de prólogo y descriptivos; finalmente el mantenimiento se debe apoyar en toda la documentación técnica y del usuario elaborada durante la elaboración del producto.

La actividad de mantenimiento generalmente la realiza el mismo grupo que desarrollo la aplicación, por lo que si es así, ellos se encuentran familiarizados con el sistema, entienden la filosofía de desarrollo y saben sin problemas el funcionamiento del sistema, sin embargo cuando se trabaja en un nuevo proyecto entonces se pueden presentar interrupciones al surgir un nuevo mantenimiento. Cuando el mantenimiento lo va a proporcionar un grupo diferente al de desarrollo, entonces se debe tener especial cuidado en los estándares y en la documentación de alta calidad, y con esto se libera al grupo de desarrollo original para atender de lleno a otros proyectos.

En muchas ocasiones existe una mejor recompensa y un mayor reto el desarrollar software nuevo que el darle mantenimiento a un software existente, por lo que la mayoría de los programadores desea desarrollar nuevas aplicaciones y no le interesa realizar mantenimientos, por lo que una forma de contrarrestar este problema puede consistir en rotar a los programadores entre el desarrollo y el mantenimiento, con esto se tiene la necesidad de hacer código de alta calidad y tener la adecuada documentación, se proporciona un mejor sentido de apreciación de las habilidades requeridas para ambas clases de programadores, se tiene mas flexibilidad en el personal y se mejora la experiencia del mismo.

Solicitud de cambios

La solicitud de cambios es el primer paso para la administración del proceso de mantenimiento y sirve para solicitar correcciones o hacer adaptaciones al sistema. Si las adaptaciones al sistema son mayores, entonces se considerarán como un proyecto nuevo. Esta solicitud es revisada por el analista y algunas veces por un comité que forma una junta de control de cambios, quienes deciden las acciones a tomar para la solicitud en proceso.

XXI.4.- Esfuerzo dedicado en el mantenimiento del código fuente

Las medidas de complejidad basadas en las propiedades del código fuente se pueden utilizar para comparar dos versiones de un programa, antes y después de una modificación y verificar si las actividades de mantenimiento están o no degradando al código original, pues si la complejidad del código fuente se incrementa con cada modificación se puede llegar a tener un código fuente inmantenible a diferencia del código estructurado y fácil de comprender que se tenía de inicio.

El esfuerzo dedicado el mantenimiento de software se divide en actividades productivas (análisis y evaluación, modificación del diseño, codificación, realización de pruebas de regresión, etc.), y en actividades menos productivas (revisar el código fuente para comprender lo que hace, tratar de interpretar las estructuras de datos, las interfaces, los límites de rendimiento, etc.)

La siguiente ecuación de Halstead mide únicamente la complejidad de código y en ella se involucran las siguientes variables :

N1 : número total de operaciones
N2 : número total de operandos
n1 : Número de operadores únicos en el programa
n2 : número de operandos únicos

El volumen del programa se determina por :

$$V = (N1 + N2)Log(n1 + n2)$$

El nivel de abstracción del lenguaje es :

$$L = \frac{2n2}{n1N2}$$

Por lo que el esfuerzo para leer y comprender un programa se da como el volumen dividido entre el nivel de complejidad (V/L).

Resumen

La falta de planeación de un proyecto es la principal causa de retrasos en la entrega de los mismos, pues a través de ella se deben de definir las actividades, tiempos y métodos que se llevarán a cabo durante la vida de todo el proyecto. Se dice muchas veces que la falta de planeación se debe al desconocimiento de los objetivos, metas y requisitos del usuario propietario de la idea de cambio, sin embargo, uno de los principales retos de la planeación, es el de aclarar las necesidades, los objetivos y las reeetricciones de todo sistema en su etapa inicial. Cualquier proyecto de desarrollo debe contar con una estructura organizacional, una de equipos y un mecanismo para asignar y evaluar el trabajo, pues de esta forma se establecen los lineamientos para empezar a trabajar de manera práctica y ordenada.

La estimación de costos y el análisis de riesgos nos dan una idea mas clara de lo que el proyecto nos costará en tiempo, dinero y esfuerzo, pues utilizando varias técnicas los ingenieros de desarrollo se basan en datos históricos, en su experiencia y en ecuaciones matemáticas, para dar una aproximación de lo que representará cada proyecto de trabajo.

Los resultados esperados en todo proyecto de desarrollo deben de tener una concordancia total con las especificaciones convenidas con los clientes, por lo que a través de un buen diseño estructural y detallado se puede empezar a construir lo que será en un futuro, la solución a nuestro problema, y esto se realizará mediante el software codificado, por lo que la codificación representa al diseño hecho realidad, pues en ella se transcriben todas las instrucciones y comandos que darán vida a nuestro sistema. Por último tenemos a la etapa de mantenimiento del software como la encargada de resolver cualquier problema o modificación al sistema entregado, por lo que una buena planeación del proyecto nos conlleva a reducir tiempos en los mantenimientos de mejora y a su vez nos evita corregir errores de interpretación en el diseño.

Para producir software de calidad es recomendable realizar la planeación del proyecto apoyados de la ingeniería de software, pues a través de técnicas y documentos de trabajo se pueden alcanzar los objetivos que todo ingeniero de desarrollo persigue en sus proyectos : producir software de calidad, incrementar su productividad y tener una satisfacción plena del trabajo que realiza.

CONCLUSIONES

Para poder realizar software de calidad los ingenieros de desarrollo deben conocer toda la teoría básica de programación (Estructuras de datos, operaciones elementales, estructuras de control, etc.) así como el de asegurar que las técnicas y métodos de programación sean los adecuados. Por otro lado, los ingenieros funcionales, reingenieros y/o los administradores de todo proyecto deben de asegurar que están empleando y administrando los mecanismos necesarios y adecuados para conseguir proyectos con calidad.

Los administradores de proyectos deben asegurar que el trabajo realizado sea hecho bajo estándares o procedimientos bien definidos, y manejar a las situaciones no controladas con cierta flexibilidad e inteligencia, también se debe tomar en cuenta la planeación estratégica del proyecto, pues en gran parte de ello dependerá el costo y el tiempo invertido.

Una vez contactado a un proyecto de desarrollo antes que nada se debe evaluar la aplicación (se deben de determinar impactos del proyecto, así como si esta de acuerdo a las políticas de la empresa), pues a partir de ello se determinará la viabilidad de éste. Posteriormente se deberá asignar a alguien o a un grupo de trabajo, para que éste elabore la "Carta de Especificaciones" acerca de lo que el cliente requiere, y se encargará(n) de elaborar y construir la aplicación de acuerdo a las especificaciones convenidas; el sistema será revisado y aprobado por parte de los desarrolladores y usuarios finales, para finalmente ser implantado en un ambiente productivo.

Durante la vida de todo proyecto se deben utilizar controles para determinar el estatus y avance de cada proyecto, por lo que las "cascadas de compromiso" se utilizan como herramientas para observar las diferentes actividades en que el proyecto se descompone, quienes son los responsables por actividad y/o etapa, en donde se identifican atrasos y o logros, etc. Por lo que representan un medio altemo para la toma de decisiones de todo proyecto de desarrollo.

ANEXOS

Anexo 2.1.- Solicitud de Desarrollo

Anexo 2.2.- Carta de Especificaciones

Anexo 2.3.- Pruebas de Integridad

Anexo 2.4.- Pruebas de Validación

Anexo 2.5.- Carta de liberación

SOLICITUD DE DESARROLLO

Folio: _____
Fecha de
Solicitud: dd/mm/aaaa

Proyecto: _____

Tipo de requerimiento:
Desarrollo Nuevo __X__ Mantenimiento de mejora _____ Mantenimiento correctivo _____

Usuario solicitante: _____
Usuario propietario del proceso: _____
Usuario responsable del área: _____

Descripción y antecedentes:

Objetivo:

Alcances:

Beneficios cuantitativos:

Beneficios cualitativos:

Documentación adjunta (Opcional):
Fotografía proceso actual
si_____ no_____
Medidores
si_____ no_____
Otros
si_____ no_____

Comentarios:

Firmas:

Usuario Solicitante | Usuario propietario del proceso | Usuario responsable del área

Guía de llenado de la Solicitud de desarrollo

- Folio: Este campo se dejará en blanco y posteriormente será llenado por los responsables del desarrollo
- Fecha: Fecha de Recepción de la Solicitud
- Proyecto: Nombre del proyecto.
- Tipo de Requerimiento
  - Desarrollo Nuevo: No existe Sistema, es un desarrollo nuevo.
  - Mantenimiento de mejora: Hacer un cambio nuevo o modificación al sistema existente.
  - Mantenimiento Correctivo: Cuando el Sistema no da los resultados esperados.

- Usuario solicitante: Datos del usuario que realiza el requerimiento.
- Usuario propietario del proceso: Datos del usuario responsable de la operación del sistema a desarrollar.
- Usuario responsable del área: Datos del responsable del área en donde operará el sistema.
  Nota: Los datos deben incluir: Nombre, correo, teléfono o ext. y área a la que pertenecen.

- Descripción y Antecedentes: Especificación general y hechos relevantes que apoyan el requerimiento.
- Objetivo: Lo que se desea alcanzar con la realización del proyecto.
- Alcances: Que áreas y/o sistemas estarán involucradas en el proyecto.
- Beneficios cuantitativos
  - Decrementar costos:
    Reducción de personal indicando número de plazas y montos.
    Eliminación de papelería, sellos, formatos especiales, etc., indicado en montos.
  - Obtener utilidades:
    Generación o incremento de utilidades por la venta de productos, especificando la proyección de ventas en relación con el mercado.
  - Ampliar gama de productos:
    Creación de productos para tener una gama más completa de éstos y poder hacer ventas cruzadas, indicando claramente cuál será la proyección de ventas en relación con el mercado
  - Responder a requerimientos oficiales:
    Cubrir un requerimiento especial regulatorio de alguna institución oficial, especificando la sanción por incumplimiento en monto.

- Beneficios cualitativos

  Mejora de productos o servicios:
  Aumentar las bondades de un producto o servicio.
  Disminución de tiempos de entrega de servicios y/o productos.
  Cubrir necesidades básicas.
  Generar la infraestructura de informática para un nuevo negocio, producto o servicio.
  Mantener competitividad en el mercado

- Documentación adjunta: En algunos casos es deseable contar con flujos de procesos, medidores y de otros documentos que nos ayuden a agilizar la etapa de análisis

- Comentarios: Aquí se registrarán todos los comentarios que surjan en relación al requerimiento en las diferentes etapas en que se encuentre incluso ambiente sugerido, fuente de información, periodicidad, nivel de servicio, volúmenes, etc..

Ejemplo Solicitud de Desarrollo

Folio: _____
Fecha de
Solicitud: 28/01/2000

Proyecto: Sistema de Cheques Comerciales

Tipo de requerimiento:
Desarrollo Nuevo __X___ Mantenimiento _____ Mantenimiento correctivo _____

Usuario solicitante: Patricia Sánchez Zurita , e-mail: PATRSZ, ext. 4729, Certificación de Adeudos
Usuario propietario del proceso: Norma Lilia Morales Reyes, e-mail: NORMMR, ext. 4690, Certificación de Adeudos
Usuario responsable del área: Roberto Carreño Lozano, e-mail: ROBECLO, ext. 4690, Certificación de Adeudos

Descripción y/o antecedentes:
Descripción: Análisis, diseño y construcción de tablas en la Base de datos actual del sistema de cheques Comerciales para el sistema de TirEmpresa accese a los datos, procesos de ventas y dotaciones, con el nivel de detalle requerido para obtener el soporte necesario de estas operaciones.

Antecedentes: Anteriormente AMIX dotaba Cheques Comerciales físicamente a las sucursales habiendo o no enviado previamente el archivo de dotaciones, y las sucursales debían, al recibir la dotación física, "activar" para su venta esa dotación; como las ventas no se realizaban en línea, frecuentemente los cheques eran vendidos sin esta reactivación, lo que generaba errores. No existía un responsable de la dotación pues el sistema no lo contemplaba; esto generaba pérdida de las mismas, a veces recuperables desde otra sucursal y a veces irrecuperables. Se realizaban las ventas por ventanilla, el cajero debía pasar por la lectora de cheques el primer cheque a vender y digitar en C.T. la cantidad de cheques a vender, así como su denominación; esto generaba un sinnúmero de errores pues los cajeros rara vez pasaban el primer cheque, por lo que las series en el sistema estaban constantemente desfasadas. Estas ventas de Cheques Comerciales eran muy lentas, aproximadamente de 25 minutos cada una, no eran comisionables y la nota de venta se llenaba manualmente lo que generaba nuevos errores. Al realizar la corrección de las ventas, el operador del sistema MUY RARA VEZ contaba con el respaldo confiable de la venta y las correcciones redundaban en errores mayores. Esto ocasionó que las ventas transmitidas a AMIX, eran incorrectas y motivaban multas por retrasos en la conciliación de las mismas.

Objetivo:
1.- Realizar la venta de Cheques Comerciales a través de un front-end (TirBital) ya conocido por las sucursales y que facilitará las ventas.
2.- Contar con la actualización del inventario en línea en el momento que se realice la venta de Cheques Comerciales a nivel nacional.
3.- Eliminar la trx de detalle de Cheques Comerciales en caja, liberando tiempo para el cajero y agilizando las filas en sucursal.
4.- Eliminar el llenado manual del formato de AMIX para cliente y ejecutivo.
5.- Contar con un acuse electrónico en sucursales sobre las dotaciones recibidas, eliminando los errores por la captura por series .
6.- Personalizar la venta de Cheques Comerciales para hacerlos comisionables.

7.- Incentivar la venta en sucursales.

Alcances:
Este sistema se utiliza en sucursales a nivel nacional para la recepción de dotación y las ventas en línea en horario de sucursal (08:00 a 20:00 hrs.); en el área de Operación Internacional (08:00 a 22:00 hrs.) para la conciliación, consulta y aclaración de las ventas realizadas y por AMIX (08:00 a 22:00 hrs.) para depositar vía host to host los archivos de las dotaciones y recibir los archivos de las solicitudes así como consultar las dotaciones aceptadas o rechazadas por las sucursales.

Beneficios cuantitativos:
- Aumentar la venta anual de cheques Comerciales de $31,871,000.00 pesos a $58,830,000.00 pesos anuales (más del 30%).
- Aumentar la venta promedio mensual por sucursal a $ 10,000.00 pesos. lo cual generará un aumento en comisiones adicionales por $411,810.00 pesos. Anuales. En 1997 la comisión anual fue de $223,097.00 pesos.

Beneficios cualitativos:
- Penetración de Bital 15.27%, posicionándolo en los primeros 5 lugares de venta de cheques Comerciales.
- Liquidación oportuna de venta de cheques al 100% a Amixco.
- Reducir los errores de venta de cheques Comerciales en sucursal, consultando y actualizando inventario en línea.
- Optimizar rutinas de conciliación y validación.
- Responsabilizar al ejecutivo de las dotaciones.

Documentación adjunta (Opcional):
Fotografía proceso actual
si__X__ no_____
Medidores
si__X__ no_____
Documento Funcional
si__X__ no_____
Otros
si_____ no__X__

Comentarios:

Firmas:

Usuario Solicitante	Usuario propietario del proceso	Usuario responsable del área

Carta de Especificaciones

La carta de especificación es elaborada por parte del productor de software. Se redacta una vez que ha recibido, por parte del cliente, la solicitud de desarrollo y toda la información relacionada con el proceso a ser automatizado. Este documento será redactado, tantas veces sea necesario, para obtener la aprobación por parte del cliente de que lo presentado ante él es lo que en su momento nos transmitió.

Procedimiento de Elaboración

1. El desarrollador deberá plasmar en varias secciones los datos del proyecto:

a) **Encabezado.** Debe contener:
   - i) Folio
   - ii) Nombre del proyecto
   - iii) Fecha de revisión
   - iv) Nombre de los revisores del documento
   - v) Número de revisión

b) **Detalle.** Es el cuerpo de la Carta de Especificaciones y estará constituido por:
   - i) Datos y/o condiciones de entrada o alimentación al sistema. Esta sección es una descripción detallada de los datos que servirán de entrada al programa: datos capturados, archivos de entrada, tablas de consulta externas, datos arrojados por otros programas, etc.. Se hará una breve descripción de la finalidad de cada entrada, su interacción con el sistema a desarrollar y su frecuencia de uso.
   - ii) Usuarios potenciales. Debe definirse quién o quienes serán los usuarios que tendrán en sus manos el producto final.
   - iii) Descripción narrativa de los procesos que efectúa el sistema. Utilizando para tal efecto una numeración consecutiva y de múltiples niveles para diferenciar un proceso de otro y sus diferentes subprocesos. Esta sección es la parte central de la Carta de especificaciones, si el usuario queda conforme con nuestra visión de su problema y como hemos de resolverlo, podemos casi asegurar que el producto final será rentable en gran medida.
   - iv) Descripción de condiciones de excepción. En esta parte se listan todas y cada una de las condiciones que tienen un particular efecto en los procesos normales. Deberá describirse la condición y el procedimiento a seguir para su tratamiento.
   - v) Fórmulas o algoritmos a utilizar por el programa. Deberán especificarse detalladamente todas las fórmulas o cálculos que se empleen dentro del programa así como el origen de los datos que utilizan. Al igual que los procesos, deberá utilizarse una numeración consecutiva y de múltiple nivel si es necesario.
   - vi) Datos y/o productos de salida. En esta sección deberán describirse las salidas que el usuario espera recibir del programa en cuestión: archivos de texto, archivos en excel, gráficas, reportes, etc. Deberá contener además los siguientes elementos:
     - a) Formatos de salida. Deberá acompañarse con un ejemplo de cada formato a utilizar, los cuales deberán tener referencia dentro de la carta de Especificaciones.
     - b) Destino de la salida. Especificación de cómo recibirá el usuario la salida del programa: archivo, pantalla; y el lugar dónde será ubicada la salida.

2. Redactada la Carta de Especificaciones, deberá ser revisada por el cliente y el desarrollador para que se efectúen las observaciones pertinentes.

3. Las observaciones deberán ser anotadas en el área del formato destinada para los resultados de la revisión, denominada **Aceptación del Producto**. Se procederá a la siguiente etapa de acuerdo a lo siguiente:

   a) Si el documento es correcto en su contenido se procede al desarrollo de la aplicación.
   b) Si el documento es en su generalidad correcto y sólo es necesario corregir algunos aspectos de mínima importancia, se procede a la corrección del documento y posteriormente al desarrollo de la aplicación. No es necesario efectuar una revisión, sólo se firma el nuevo documento.
   c) Si el documento no concuerda en algunos puntos con lo transmitido por el cliente, deberá reelaborarse la Carta de Especificación con las correcciones necesarias y deberá presentarse para una nueva revisión.

4. En todos los casos anteriores la Carta de Especificaciones, deberá ser firmada por el encargado de desarrollo, el líder del proyecto, el propietario del proceso y el responsable del área para validar el acuerdo. El dueño del proceso deberá conservar una copia para futuras aclaraciones.

5. Finalmente, La Carta de Especificaciones será anexada a la documentación del proyecto.

Pruebas de Integridad

El material de trabajo de las pruebas de Integridad es sin lugar a dudas la Interfaz de Usuario. En base a ello, lo que debe cuidarse es el cómo se comporta el programa ante el cliente, debe cuidarse todo aquello con lo que tenga contacto.

Elementos Básicos de Integridad

A continuación se listan los elementos que son viables de pruebas de integridad:

1. Formatos de campos. Deben corroborarse que todo dato de entrada o de consulta utiliza los formatos de presentación previamente establecidos. Por ejemplo, toda fecha debe ser utilizada en el mismo formato "dd/mm/aaaa".
2. Estandarización en entradas. Es una sana práctica de desarrollo estandarizar las entradas de datos en las pantallas de captura; esto es, definir si todas las entradas serán capturadas en mayúsculas, minúsculas o libremente.
3. Inicialización de campos con valores por omisión. debe facilitársele el trabajo en todo momento al usuario. Si es posible y no causa problemas de operación, siempre que sea posible hay que dar valores a los campos de captura en forma automática.
4. Indicación de campos en captura por el usuario. A menudo, y a pesar de que se ha hecho todo lo posible por simplificar, las pantallas de captura cuentan con gran cantidad de datos. En estos casos es conveniente idear algún medio para indicar al usuario cual es el campo activo en el momento en que el está capturando.
5. Verificar entradas requeridas. Es un error de diseño grave permitir que un programa se ejecute aún cuando hay valores necesarios que no fueron proporcionados por el usuario. Obviamente esto causa errores y disgustos y debe evitarse.
6. Validación de tipo de Datos. Todo dato capturado debe corresponder al tipo de dato para el que fue creado. No es posible aceptar letras donde inicialmente se ha pensado aceptar números.
7. Presentación inteligente de los campos de captura. El usuario no tiene porque entrar o ver campos que no correspondan a la acción que en el momento esté realizando.
8. Facilidad de asimilación de la información mostrada. Debemos hacer fácil el rastreo y consulta de datos al cliente. No debemos crear aplicaciones que muestren información utilizando pantallas y más pantallas. El usuario no experimentado, fácilmente se confunde.

9. Facilidad de selección de opciones. Cuando haya que elegir entre varias opciones, el usuario debe hacer el menor esfuerzo en seleccionar la opción deseada.
10. Orden de elementos mostrados en pantalla. La entrada a los campos de captura y a los elementos de interacción con el usuario debe seguir un orden lógico que de agilidad al manejo de la información.
11. Disponibilidad de ayuda. Un sistema debe proporcionar por lo menos el mínimo de ayuda a los usuarios. La finalidad de la ayuda, además de apoyar al usuario en momentos de contingencia, es la de guiar en todo momento al usuario y eliminar posibles errores.
12. Confirmación antes de cualquier acción de impacto. Toda acción de consecuencias irreversibles debe ser confirmada antes de efectuarse.
13. Herramientas de respaldo y recuperación de la Información. Debe pensarse que también es humano y puede equivocarse. Por ello todo sistema debe implementar algún sistema de respaldo y recuperación de datos.
14. Condiciones de validación especial. Es necesario asegurarse de que todo dato introducido por el usuario es el correcto para el contexto en el cual se solicita (p. ej. la fecha de matrimonio de una persona no puede ser menor a su fecha de nacimiento).

Dado que las aplicaciones varían dependiendo del problema que traten de resolver, los puntos anteriores deben ser cubiertos de acuerdo a las características de la aplicación en cuestión. No es conveniente tratar de definir una matriz o lista maestra que se aplique a cualquier aplicación en desarrollo. Además que esto es muy complicado, no es rentable, ya que muchas características que aplican en un caso en otro son obsoletas.

Para cada nuevo tipo de software debe crearse una matriz de verificación especial que cubra al menos con los elementos básicos expuestos.

Procedimiento de Aplicación

1. Definir una lista de verificación en base a los puntos básicos de integridad discutidos.

2. Una vez construida la lista de verificación, esta debe llenada de acuerdo a lo siguiente:

   a) **Encabezado**. Debe contener:
      i) Folio
      ii) Nombre del proyecto
      iii) Fecha de revisión
      iv) Nombre del ejecutor de las pruebas.

v) Número de revisión número consecutivo que señala las veces en que se han ejecutado las pruebas de integridad sobre el software, o bien, señala el cambio de lista por modificaciones a su estructura.

b) **Detalle.**

i) Cada una de las pruebas especificadas en la lista de verificación debe ser realizada sin excepción. Si se detectan errores antes de concluir las pruebas debe continuarse hasta terminar con la lista. Las pruebas se realizan de dos formas:

a) Prueba efectuada por el desarrollador del software.

b) Prueba efectuada por el usuario final. En esta prueba el desarrollado acompaña al usuario para efectuar la prueba.

ii) El resultado de cada prueba solo puede tener uno de tres valores: Aprobada, No aprobada o No Aplica. No son válidos los resultados intermedios.

3. Una vez concluidas las pruebas, se registra la evaluación global en el apartado de **Resultados de la Prueba.** Se procede a la siguiente etapa de acuerdo a lo siguiente:

a) Si todas las pruebas fueron aprobadas con éxito se continúa con la siguiente fase del proyecto.

b) Si al menos una prueba de integridad no es aprobada, no es posible continuar con la siguiente fase del proyecto. El desarrollador debe corregir los errores u omisiones detectados y volver a iniciar el proceso de Pruebas de Integridad tantas veces como sea necesario.

4. En ambos casos la Lista de Verificación deberá ser firmada por el encargado de desarrollo, el propietario del proceso y el responsable del área para validar el acuerdo. El dueño del proceso deberá conservar una copia para futuras aclaraciones.

5. Finalmente, el documento resultante será anexado al archivo del proyecto.

Pruebas de Validación

Las pruebas de validación representan la funcionalidad del sistema, pues en él se reflejan las condiciones o requerimientos que nuestro cliente solicitó.

Procedimiento de Aplicación

1. Se debe determinar en conjunto con el usuario la serie de condiciones y variables críticas que pueden presentarse durante la ejecución de la aplicación, así como las condiciones de seguridad, resistencia y tensión para sistemas multiusuario.
2. Se elabora una matriz donde se refleje las posibles combinaciones entre las variables y condiciones definidas. Se marcan los cruces válidos. Estos cruces constituyen las condiciones que es necesario probar.
   En algunos casos será necesario limitar el número de pruebas a las que se puede someter el software. Es posible que el numero de combinaciones entre las condiciones críticas sea en extremo grande como para probar cada una de ellas. En este punto, tanto desarrollador como usuario, deberán acordar las pruebas que se consideren "*Suficientes y Necesarias*" como para aprobar el software. El resto de las combinaciones no probadas se asumen como válidas siempre y cuando estén dentro de alguna de las pruebas maestras.
   Además, existirán algunas condiciones que será necesario verificar en cada una de las pruebas. A este tipo de condiciones o variables las llamaremos *Validaciones Requeridas*, y serán colocadas en la parte inferior de la matriz para tener referencia de ellas. El resultado de verificar este tipo de condiciones será anotado en una columna especial de la matriz denominada de igual forma: Validaciones Requeridas.

   La matriz será la lista de verificación que, de igual forma que las pruebas e integridad, se llevan a cabo entre el desarrollador y el usúario. Para su redacción se utilizan los siguientes puntos como sigue:

a) **Encabezado**. Debe contener:
   - i) Folio
   - ii) Nombre del proyecto
   - iii) Nombre del módulo o subsistema
   - iv) Fechas de revisión, (inicial y final)
   - v) Nombre del Revisor
   - vi) Número de revisión número consecutivo que señala las veces en que se han ejecutado las pruebas de validación
   - vii) Número total de Páginas

b) **Detalle.**
   - i) Cada una de las pruebas especificadas en la lista de verificación debe ser realizada sin excepción. Si se detectan errores antes de concluir las pruebas, a diferencia de las pruebas de integridad, debe detenerse el proceso.
     - a) Prueba Alfa: Prueba efectuada por un desarrollador distinto al que programó el sistema de software.
     - b) Prueba Beta: prueba efectuada por el usuario final.

ii) El resultado de cada prueba solo puede tener uno de dos valores: Aprobada, No aprobada.

iii) Los resultados de toda prueba: archivos, reportes, consultas en pantallas, etc; deberán almacenarse en físico o electrónicamente, junto con su correspondiente formato.

3. Una vez concluidas las pruebas, o bien, si fueron detenidas por fallo en alguna prueba, se registra la evaluación global en el apartado de **Resultados de la Prueba.** Se procede a la siguiente etapa de acuerdo a lo siguiente:
   a) Si todas las pruebas fueron aprobadas con éxito se procede con la Liberación del Sistema.
   b) Si una prueba de validación no es aprobada, no es posible continuar con la siguiente fase del proyecto. El desarrollador debe corregir los errores u omisiones detectados y volver a "iniciar" el proceso de Pruebas de Validación tantas veces como sea necesario.

4. En ambos casos el documento deberá ser firmado por el encargado del desarrollo y el revisor de las pruebas (Usuario final y/o Desarrollador revisor). El dueño del proceso deberá conservar una copia para futuras aclaraciones.

5. Finalmente, el documento resultante será anexado al archivo del proyecto.

Carta de liberación

Datos del proyecto:
Folio:
Nombre:
Fecha de liberación:

Por medio de la presente se firma el acuerdo entre el cliente y el suministrador, en el cual confirman y aceptan que el producto cumple con las especificaciones descritas en la propuesta de prestación de servicios, por lo que a partir de esta fecha, cualquier modificación distinta a las ya convenidas, será sujeta a un nuevo estudio y se le asignará una nueva prioridad (Excepto aquellas situaciones en que por responsabilidad del suministradoir, el sistema tenga fallos, en este caso la modificación se realizará en forma inmediata).

Nombres y firmas de aceptación

Cliente(s):

_____

Sumministradore(s):

_____

♣ BIBLIOGRAFÍA

Nombre : Computación e Informática Hoy
Computer Currents Navigatings Tomorrow's Technology
Autor : Goerge Beekman
Editorial : Iberoamericana
País : USA
Año : 1995

Nombre : Ingeniería de Software
Autor : Roger S. Pressman
Editorial : McGraw-Hill
País : Madrid
Año : 1993

Nombre : Ingeniería de Software. Temas Selectos de Ingeniería
Autor : D. C. Ince
Editorial : Iberoamericana
País : México
Año : 1993

Nombre : Programación estructurada
Autor : Leobardo Lopez R.
Editorial : Computec Editores
País : México
Año : 1994

Nombre : Sistemas de Información para la Administración
Autor : James A. Senn
Editorial : Trillas
País : México
Año : 1991

Nombre : Análisis y Diseño de Sistemas
Autor : kenneth E. Kendall y Julie E. Kendall
Editorial : Prentice Hall
País : México
Año : 1991

Nombre : Conceptos de informática
Autor : Peter Bishop
Editorial : Anaya Multimedia, S.A.
País : Madrid
Año : 1989
Pag. : 586

Nombre : Diseño de Bases de Datos
Autor : Gio Wiederhold
Editorial : Mc Graw-Hill
País : México
Año : 1985

Nombre : Principios de Sistemas de Información
Autor : George M : Scott
Editorial : Mc Graw-Hill
País : México
Año : 1985

Nombre : Técnicas de bases de datos
Autor : Shakuntala Atre
Editorial : Trillas
País : México
Año : 1991

Nombre : Computadoras y Sistemas de Información en los Negocios
Autor : Goerge J. Brabb
Editorial : Interamericana
País : México
Año : 1983

Nombre : Metodología de la programación
Autor : Edualdo Alcalde y Miguel García
Editorial : McGraw-Hill
País : México
Año : 1990

Nombre : Metodología de la programación Estructurada
Autor : María Dolores Alonso y Silvia Rumeo
Editorial : Paraninfo
País : España
Año : 1992

Nombre : Ingeniería de Software
Autor : M. Thorin
Editorial : McGraw-Hill
País : México
Año : 1990

Nombre : Análisis y diseño de Sistemas de Información
Autor : Senn
Editorial : McGraw-Hill
País : México
Año : 1990