Universidad Nacional Autónoma de México

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"A R A G O N"

"INGENIERIA EN PROGRAMACION"

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A N

GUSTAVO MAYEN FLORES

ADRIAN PAREDES CHAVEZ

SAN JUAN DE ARAGON, MEX. 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A Dios:

A quien le agradecemos el habernos dado vida y salud, para realizar nuestros objetivos.

A Nuestros Padres:

Sra. Cristina Flores M.

Sr. Agustin Paredes Cardona y Sra. Luz Chávez G.

Les damos gracias por el esfuerzo y sacrificio que otorgaron en nuestra educación, por ese gran apoyo incondicional que nos han brindado y que por ello hoy vemos realizada nuestra meta. Por ese gran amor y muchas otras cosas...

...Muchas Gracias

A Nuestras Novias:

Ma. de los Angeles Hernandez L.

Guadalupe Mayen F.

Por el gran apoyo y aliento que nos brindan en los momentos difíciles, y por que en ustedes encontramos la motivación para seguir adelante.

A Nuestros Hermanos:

Alma, Angelica, Arturo, Armando, Margarita, Natividad, Guadalupe, Virginia, Gabriela, Gonzalo, Fernando.

La familia es el cimiento más fuerte sobre el cual construimos el amor y la seguridad. Gracias por su cariño, amor y comprensión y sobre todo por su desinteresada ayuda.

A Nuestros Profesores:

Que nos enseñaron el camino del conocimiento y de la reflexión, y a nuestro asesor David González Maxínez, por su apoyo en la elaboración de este trabajo.

A TODAS AQUELLAS PERSONAS QUE DE UNA U OTRA MANERA HAN COLABORADO EN LA REALIZACION DE ESTE TRABAJO.

INDICE

MARCO TEORICO

A finales de la decada de los 70's y principios de los 80's la tecnología en México a tenido una evolución muy importante, donde la computadora juega un papel muy especial.

A partir de ese momento se presento un fenómeno muy curioso, debido a que la gente no sabía que hacer y como explotar las computadoras.

Es por eso que la Universidad Nacional Autonoma de México al percatarse de este problema creó en 1978 la carrera de Ingeniería en Computacón (impartida por la Facultad de Ingeniería en Ciudad Universitaria) con el objetivo de crear especialistas en esta área.

Fue tanta la demanda con respecto a la carrera de Ingeniería en Computación, que la UNAM decidió en 1981 implantarla en la Escuela Nacional de Estudios Profesionales Aragón.

Aquí en la ENEP Aragón, la carrera se implantó como complemento de las carreras de Ingeniería Mecánica Electrica, Ingeniería Industrial, etc; por lo que el programa contenía muy pocas materias sobre el área de computación, pero que era suficiente para las necesidades de esa época.

Debido al gran desarrollo tecnologico que ocurre día con día, y al cambio de las necesidades en el campo de la computación, se observó que el plan de estudios vigente no cumplía con las necesidades actuales es este ramo, por lo que se decidió cambiarlo, para que el egresado de esta carrera cumpliera con el perfil necesario que se requería en la actualidad.

Dentro del nuevo plan de estudios, se hizo principalmente una clsificación de asignaturas, separandolas de la siguiente manera:

- Básicas (Tronco común).
- Metodologicas.
- Específicas.

Donde se modificaron algunos programas de asignaturas ya existentes y se implantaron otras.

En el área de software, se observó una deficiencia en el proceso de elaboración de software de aplicación por lo que se optó por implementar la asignatura de Ingeniería de Programación.

En esta materia se tiene como objetivo principal, que el alumno conozca y ponga en uso los aspectos de planeación, administración del proyecto, el uso de técnicas estructuradas, la documentación y puesta en marcha del sistema.

Dicha materia está ubicada dentro de las materias específicas de la carrera, impartida en el 6º semestre, como se muestra en la figura y esta compuesta por el siguiente programa:

Clave: 0407

Nº de Creditos: 08

Semestre: 6º

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGON

PLAN DE ESTUDIO PROPUESTO PARA LA CARRERA DE INGENIERIA EN COMPUTACION

AREA DE CONOCI.	B A S I C A S					SOCIO	E S P E C I F I C A S D E L A C A R R E R A										CREDITOS
	MATEMATICAS			METODOLOGIA		HUMANISTIC.	SOFTWARE		HARDWARE		SISTEMAS		ELECTRICA	ELECTRONICA	COMUNICACI.	CONTROL	
1	ALGEBRA (8)	GEOMETRIA ANALITICA (8)	CALCULO DIFERENCIAL E INTEGRAL (8)	COMPUTAC. Y PROGRAM. (8)	INTRODUCC. A LA INGENIERIA												48
2	ALGEBRA LINEAL (8)	CALCULO VECTORIAL (8)				OPTATIVA DE HUMANID. (8)	MANEJ ESTRU Y CARAC DE LENGUAJES (8)				MICROPR. CONTROLES Y COMTAB (8)						37
3	METODOS NUMERICOS (8)	ECUACIONES DIFERENCIA.				INTRODUC. A LA ECONOMIA (8)	ESTRUCTURA DE DATOS (8)						ELECTRICIDAD Y MAGNETISMO (12)				48
4	PROBAB. Y ESTADISTI. (8)						ESTRUCTURA DISCRETA (8)				TECNICAS DE EVALUAC. ECONOMICA (8)	INGENIERIA DE SISTEMAS (8)				ANALISIS DE SISTEMAS Y SEÑALES (8)	53
5							L. FORMALES Y AUTOMATA (8)				PROGRAMAC. DE SISTEMAS (8)		ANALISIS DE CIRCUITOS ELECTRICOS (12)	DISPOSITIV. ELECTRONIC. (12)		DISEÑO DE SISTEMAS FISICOS (8)	64
6							INGENIERIA DE PROGRAMAC. (8)	SISTEMAS OPERATIVOS (8)	DISEÑO LOGICO (10)		INVESTIG. DE OPERACIONES (10)					MEDICION E INSTRUMT. (10)	48
7							BASES DE DATOS (8)		MEMORIAS Y PERIFERICOS (8)	DISEÑO DE SISTEMAS DIGITALES (10)					FILTROS Y MODULACION (10)	CONTROL ANALOGICO (10)	48
8							SISTEMAS DE INFORMACION (8)	COMPILA-DORES (8)	ORGANIZACI. DE COMPUTADORAS (8)						COMUNICACI. DIGITALES (10)	CONTROL DIGITAL (10)	64
9							INTELIGEN. ARTIFICIAL (8)		MICRO-COMPUTADORAS (10)		ORGANIZACI. Y DISEÑO DE COMPUTO (8)	ESCOGER DOS MATERIAS DE LA LISTA DE OPTATIVAS QUE SE OFRECEN					48
10						ACTIVOS Y NECESIDADES DE MEXICO (8)	AREA DE COMPUTACION REDES DE COMPUTADORES SEMINARIO DE ING. COMP. (8) (8)				ESCOGER DOS MATERIAS DE LA LISTA DE OPTATIVAS QUE SE OFRECEN						24

OPTATIVAS DE HUMANIDADES

A escoger una de las siguientes opciones:

	AREA	CREDITOS
- Comunicacion oral y escrita	C	8
- Sociologia	C	8
- Tecnicas del aprendizaje y la diversion	C	8
- Problemas Interpersonales y comunicacion	C	8
	C	8
	C	8

OPTATIVAS PARA 9o. Y 10o. SEMESTRE

	AREA	CREDITOS	
Reconocimiento de patrones	C	8	Diseño asistido por computadora
Procesamiento digital de lenguaje	C	10	Robotica
Procesamiento digital de Señales	C	8	Sistemas expertos
Inteligencia	SI	8	Calidad
Graficacion por computadora	CH	8	Electronica analogica
Temas especiales de computacion	C	8	Proyecto escuela-industria

TEMAS

I. Evolución de los sistemas de programación.

II. Estudio General del sistema.

III. Planeación del sistema de programación.

IV. Analisis y especificación estructurada.

V. Diseño estructurado.

VI. Codificación y los lenguajes de programación.

VII. Documentación.

VIII. Prueba y confiabilidad de los sistemas.

IX. Instalación, mantenimiento y aseguramiento de la calida de los sistemas.

OBJETIVO DEL CURSO:

El alumno desarrollará un producto de programación considerando los aspectos de planeación y administración del proyecto, el uso de las técnicas estructuradas, la documentación y puesta en marcha del sistema.

CAPITULO I.

EVOLUCION DE LOS SISTEMAS DE PROGRAMACION

INTRODUCCIÓN

Durante las tres primeras décadas de la Informática, el principal desafío era desarrollar el hardware de las computadoras de forma que se redujera el costo de procesamiento y almacenamiento de datos. A lo largo de la década de los 80's, los avances en microelectrónica han dado como resultado una mayor potencia de cálculo a la vez que una reducción del costo.

Hoy el problema es diferente, ya que el principal desafío es reducir el costo y mejorar la calidad de las soluciones basadas en computadoras.

Durante *la primera era* de las computadoras el hardware sufrió continuos cambios, mientras el software se veía simplemente como un añadido, ya que el desarrollo de software se realizaba virtualmente sin ninguna planificación. Durante este período se utilizaba en la mayoría de los sistemas una orientación por lotes.

En los primeros años la producción de software para ser vendido, estaba en sus inicios, debido a que el software se diseñaba a medida para cada aplicación y tenia una distribución sumamente pequeña. La mayoría del software se desarrollaba y utilizaba por la misma persona u organización. Es decir la misma persona lo escribía, lo ejecutaba y si fallaba lo depuraba. Era una programación muy personalizada y la documentación básicamente no existía.

La Segunda Era, se extiende desde la mitad de la década de los 60's hasta finales de los 70's. Esta era, se distingue por la aplicación de la multiprogramación y los sistemas multiusuarios, ya que estos introdujeron nuevos conceptos de interacción hombre-máquina.

Los sistemas de tiempo real podían recoger, analizar y transformar datos de múltiples fuentes, controlando así los procesos y produciendo salidas en milisegundos en vez de minutos. Los avances en los dispositivos de almacenamiento en línea condujeron a la primera generación de sistemas de gestión de base de datos.

La segunda era se caracterizó también por el uso del software como producto, ya que se desarrollaba para una amplia distribución en un mercado multidisciplinario. Los programas se distribuían para computadoras grandes y minicomputadoras a cientos y a veces a miles de usuarios. La importación de productos de software, aumento el número de sistemas informáticos así como una gran variedad de sentencias fuente.

Con todo esto apareció un gran problema, todos estos programas tenían que ser corregidos cuando se detectaran errores, cambios en los requerimientos del usuario o bien adaptaciones al nuevo hardware. Estas actividades se llamaron conjuntamente, "mantenimiento del software". Esto comenzó a absorber recursos en una medida alarmante, aun peor la naturaleza personalizada de muchos programas los hacia virtualmente imposibles de mantener, había comenzado la crisis del software.

La Tercera Era de la evolución de los sistemas de computadoras comenzó a mediados de los 70's y continúa hoy. Aquí aparecen los sistemas distribuidos (computadoras múltiples, cada una ejecutando funciones concurrentemente y comunicándose con alguna otra), Redes de área local y global, comunicaciones digitales de alto ancho de banda y la creciente demanda de acceso "instantáneo" a los datos, supusieron una fuerte presión sobre los desarrolladores de software. Esta era se caracteriza también por la llegada de los microprocesadores y las computadoras personales.

Las computadoras personales fueron la base para el crecimiento de muchas compañías de software. El hardware de las computadoras personales, se convierte rápidamente en un producto estandard, mientras el software suministrado con el mismo, marca la diferencia. Es decir, mientras que las ventas de computadoras personales se estabiliza a mitad de los 80's, las ventas de productos de software continuaron creciendo.

La cuarta era está ahora empezando. Ya las técnicas de la cuarta generación para el desarrollo de software están cambiando la forma en que algunos segmentos de la comunidad informática construye los programas de computadora. Los sistemas expertos y el software de inteligencia artificial finalmente se ha transladado del laboratorio a aplicaciones prácticas, en un amplio rango de problemas del mundo real.

I.1. LA CRISIS DE LOS SISTEMAS DE PROGRAMACIÓN

El término "Ingeniería de Programación" se introdujo por primera vez a finales de la década de 1960 en una conferencia celebrada para analizar la llamada "crisis del software". Esta crisis fue el resultado directo de la aparición del hardware de computadoras de la tercera generación, estas máquinas eran de una capacidad muy superior a la de las máquinas más potentes de la segunda generación, y su potencia hizo posibles las

aplicaciones que hasta ese momento eran irrealizables. El desarrollo de esas aplicaciones requirió la construcción de grandes sistemas de software.

La crisis de los sistemas de programación, o crisis del software no sólo se refiere a un conjunto de problemas encontrados en el desarrollo de programas de computadora. Sino que también abarca los problemas asociados de cómo desarrollarlos, cómo mantenerlos y cómo podemos esperar satisfacer la demanda creciente de desarrollo de software.

I.1.1. Problemas

Las primeras experiencias en la construcción de grandes sistemas de software mostraron que las metodologías de desarrollo hasta entonces existentes eran inadecuadas. Varios proyectos se retrasaron, costaron mucho más de lo previsto en principio y resultaron poco confiables, difíciles de mantener y de un rendimiento muy pobre. El desarrollo de software estaba en una situación de crisis.

La crisis del software se caracterizó por muchos problemas pero los responsables en el desarrollo de sistemas de programación se concentran en los aspectos más importantes como son:

- La planificación y estimación del costo, el cual frecuentemente es muy impreciso.
- La "productividad" de la gente del software no corresponde con la demanda de sus servicios.
- La calidad del software no llegó a ser a veces ni adecuada.

Estos problemas son sólo las manifestaciones más visibles de muchas otras dificultades del software como son:

- No tener tiempo para recoger datos sobre el proceso de desarrollo de software. Sin datos históricos, la estimación no era buena y por lo consiguiente los resultados eran muy pobres.
- Con frecuencia no se realizaban las expectativas del cliente, debido a la escasa comunicación entre el cliente y el desarrollador de software.
- Con todo ésto el software era difícil de mantener, ya que resultaba muy costoso.

Durante la crisis del software los costos del hardware bajaban, mientras que los del software aumentaban con rapidez. Había una urgente necesidad de nuevas técnicas y metodologías que permitieran controlar la complejidad inherente a los grandes sistemas de software.

I.1.2. Causas

Los problemas asociados con la crisis del software se han producido por el carácter del propio sistema a desarrollar y por los errores de las personas encargadas del desarrollo. El software es un elemento lógico, por lo tanto el éxito se mide por la calidad de una única entidad, si se encuentran errores, existe una alta probabilidad de que se introdujeran inadvertidamente durante el desarrollo y no se detectaran durante las pruebas. Estos errores se reemplazan durante el mantenimiento, el cual incluye la corrección o modificación del diseño.

El desafío intelectual en el desarrollo de sistemas de programación es seguramente una de las causas de esta crisis, pero los problemas tratados anteriormente han sido causados por errores humanos más sencillos.

Otra de las causas de la crisis del software es que los trabajadores han tenido muy poco entrenamiento formal en las nuevas técnicas de desarrollo de sistemas de programación. Así que cada individuo enfoca su tarea de "escribir programas" con la experiencia obtenida en trabajos anteriores, de esta manera algunas personas desarrollan un método ordenado y eficiente para desarrollar software (mediante prueba y error), pero muchos otros desarrollan malos hábitos que dan como resultado una baja calidad y muchos problemas en el mantenimiento de los programas.

Todos nos resistimos al cambio. Sin embargo, mientras el potencial de cálculo (hardware) experimenta enormes cambios, la gente del software, responsables de aprovechar dicho potencial, se oponen normalmente a los cambios cuando se discuten, y se resisten al cambio cuando se introduce. Puede que ésta sea la causa real de la crisis del software.

I.2. EL CICLO DE VIDA DE LOS SISTEMAS DE PROGRAMACIÓN.

El ciclo de obtención de sistemas o ciclo de vida de los sistemas es un enfoque por etapas de análisis y de diseño, que postula que el desarrollo de los sistemas mejora cuando existe un ciclo específico de actividades del analista y de los usuarios.

Los analistas no están de acuerdo respecto al número exacto de etapas que conforman el ciclo de desarrollo de los sistemas; sin embargo, por lo general se reconoce la importancia de su enfoque sistémico.

I.2.1. Paradigmas de la ingeniería de Programación.[1]

La crisis del software no desaparecerá de la noche a la mañana, por lo tanto el conocimiento de los problemas y sus causas, son los primeros pasos para obtener soluciones que proporcionen ayuda al desarrollador de software, y así obtener software de mejor calidad.

No existe un método que solucione la crisis del software. Sin embargo, puede lograrse una disciplina para su desarrollo, disciplina llamada ingeniería de programación.

I.2.1.1. Ingeniería de Programación.

La *Ingeniería de Programación*[2] es una disciplina que abarca un conjunto de tres elementos: métodos, herramientas y procedimientos, que facilitan al gestor controlar el proceso de obtención de software y proporcionar a los que practiquen dicha ingeniería las bases para construir software de alta calidad de una forma productiva.

Los métodos de la ingeniería de programación, abarcan un amplio espectro de tareas que incluyen: planificación y estimación de proyectos; análisis de los requerimientos del software; diseño de estructuras de datos, arquitectura de programas y procedimientos algorítmicos; codificación; prueba y mantenimiento.

Las herramientas de la ingeniería de programación proporcionan un soporte automático o semiautomático para los métodos, ya que hoy existen herramientas para cada uno de los métodos mencionados anteriormente.

1. Paradigmas tomados de "Ingeniería del Software", Roger S. Pressman.

2. "El establecimiento y uso de principios de ingeniería, orientados a obtener económicamente software que sea fiable y funcione eficientemente sobre maquinas reales", definición dada por Fritz Bauer en la primera conferencia sobre el tema.

Cuando las herramientas se integran, de forma que la información creada por una herramienta pueda ser utilizada por otra, se establece un sistema para el soporte del desarrollo de software, llamado Ingeniería del Software Asistido por Computadora (CASE: Computer-Aided Software Engineering). CASE combina el software, hardware y bases de datos de la ingeniería del software, para crear un entorno de ingeniería análogo al diseño/ingeniería asistida por computadora, CAD/CAE (Computer-Aided Design/Engineering), para el hardware.

Los procedimientos son la unión entre métodos y herramientas para facilitar el desarrollo racional y oportuno del software de computadora. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas de informes que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios, y las guías que facilitan a los gestores de software establecer su desarrollo.

La ingeniería del software esta compuesta de etapas que abarcan los métodos, herramientas y procedimientos, estas etapas se denominan paradigmas de la ingeniería del software. Un paradigma se elige basándose en la naturaleza del proyecto y de la aplicación, los métodos y herramientas a usar y los controles y entregas requeridos. A continuación se presentan tres paradigmas para la ingeniería del software.

I.2.1.2. El ciclo de vida clásico.

Aquí dividimos el ciclo en seis etapas, como se muestra en la Fig. I.1. Aunque cada etapa se presenta de manera discreta, nunca se lleva a cabo como un elemento independiente. En lugar de ello, se realizan al mismo tiempo diversas actividades, y éstas llegan a repetirse. Por ello es de mayor utilidad suponer que el ciclo de desarrollo de los sistemas transcurre en etapas y no como elementos separados.

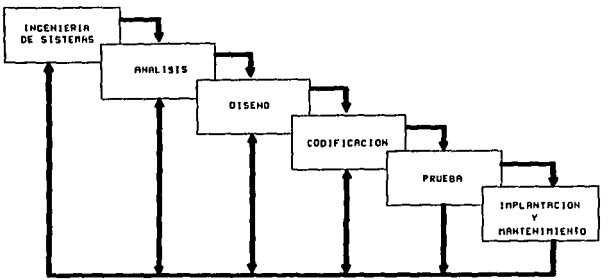


Fig. 1.1. El ciclo de vida clásico.

El ciclo de vida clásico para el desarrollo de sistemas consiste en las siguientes actividades:

- **Ingeniería y análisis del sistema.**

Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requerimientos de todos los elementos del sistema. Esta visión del sistema es esencial cuando el software debe interrelacionarse con otros elementos tales como hardware, usuarios y bases de datos.

La identificación de objetivos también es un componente importante de la primera fase. En primera instancia, el analista deberá descubrir lo que la empresa intenta realizar. Y luego, estará en posibilidad de determinar si el uso de los sistemas de programación apoyarán a la empresa para alcanzar sus metas.

- **Determinación de los requerimientos del software.**

En esta etapa el analista de sistemas determina las necesidades propias del sistema. Existen herramientas y técnicas especiales que facilitan al analista la realización de las determinaciones requeridas. Estas incluyen el uso de los diagramas de flujo de datos que representan en forma estructurada y gráfica la entrada de datos de la empresa, los procesos y la salida de la información. A partir del diagrama de flujo de datos se desarrolla un diccionario de datos que contiene todos los datos que utiliza el sistema.

- **Diseño.**

En esta etapa del ciclo de vida de los sistemas, el analista usa la información recolectada con anterioridad y elabora el diseño lógico del sistema de programación. También diseña accesos efectivos al sistema, mediante el uso de las técnicas de diseño de pantallas y formas.

- **Codificación.**

En esta etapa del ciclo de desarrollo de sistemas el analista trabaja con los programadores para construir los programas de computadora necesarios. Dentro de las principales técnicas estructuradas para el diseño y documentación del software se tienen: el método HIPO, los diagramas de flujo, los diagramas de Nassi-Schneiderman, los diagramas de Warnier-Orr y el pseudocódigo.

Durante esta etapa, el analista también colabora con los usuarios para desarrollar la documentación indispensable del software, incluyendo los manuales de procedimientos.

- **Prueba.**

El sistema de programación debe ser probado antes de utilizarlo, el programador realiza algunas pruebas por su cuenta, otras se llevan a cabo en colaboración con el analista de sistemas y otras con el cliente. En un principio, se hace una serie de pruebas, para identificar las posibles fallas del sistema.

El mantenimiento del sistema y de su documentación empiezan en esta etapa; y después, esta función se realizará de forma rutinaria a lo largo de toda la vida del sistema.

- **Implantación y mantenimiento del sistema.**

En esta última etapa del desarrollo del sistema, el analista ayuda a implantar el sistema de programación. Esto incluye la capacitación que el usuario necesita. Parte de esta capacitación la dan las casas comerciales. La supervisión de la capacitación es responsabilidad del analista de sistemas. El mantenimiento del sistema de programación, comienza en esta etapa y continuará de forma rutinaria a lo largo de toda la vida del sistema.

I.2.1.3. Construcción de prototipos.

La construcción del prototipo es un proceso que facilita al programador la creación de un modelo del software a realizar. El modelo tomará una de las tres formas siguientes: un prototipo en papel el cual mostrará la interacción hombre-máquina para facilitar al usuario la comprensión de como se producirá tal interacción; un prototipo que funcione, este implementa algunos subconjuntos de la función requerida al software deseado; o bien un programa existente que realice parte o toda la función deseada.

La fig. I.2. muestra el paradigma de la construcción de prototipos.

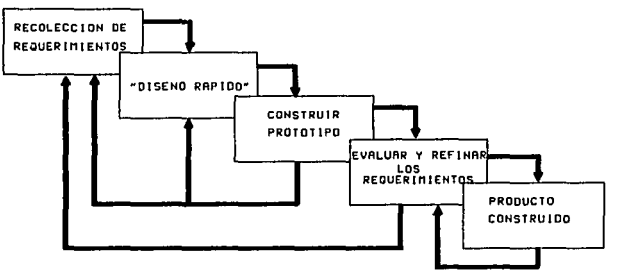


Fig. I.2. Construcción de prototipos.

Como se observa en la Fig. I.2. la construcción de prototipos comienza con la recolección de los requerimientos. El técnico y el cliente se reunen y definen los objetivos globales para el software, identifican todos los requerimientos conocidos para producir un "diseño rápido" el cual se enfoca sobre la representación de los aspectos del software, visibles al usuario, por ejemplo, métodos de entrada y formatos de salida. El diseño rápido conduce a la construcción de un prototipo, el cual es evaluado por el cliente/usuario y se utiliza para refinar los requerimientos del software a desarrollar. Para la construcción de prototipos el realizador intenta hacer uso de fragmentos de programas existentes o aplica herramientas (por ejemplo, generadores de informes, gestores de ventanas, etc.) que facilitan la rápida generación de programas.

El prototipo puede servir como "el primer sistema", pero esto no es recomendable. Como en el ciclo de vida clásico, la construcción de prototipos como paradigma de la ingeniería de programación, puede ser problemático por las siguientes razones:

- El cliente ve funcionando lo que parece ser una versión del software, ignorando que el prototipo fue construido con "chicle y alambre", ignora que por las prisas en hacer que funcione, no se consideraron los aspectos de calidad y mantenimiento a largo plazo del software.

- El técnico se compromete frecuentemente en obtener un prototipo que funcione rápidamente. Por esta razón se utiliza un sistema operativo o un lenguaje de programación inapropiado, simplemente porque está disponible y es conocido; un algoritmo ineficiente puede implementarse de forma sencilla para demostrar su capacidad.

Aunque pueden presentarse problemas, la construcción de prototipos es un paradigma efectivo, la clave está en definir al comienzo las reglas del juego; esto es, el cliente y el técnico deben estar de acuerdo en que el prototipo se construya para servir sólo como un mecanismo de definición de los requerimientos.

I.2.1.4. Técnicas de la cuarta generación.

El término técnicas de la cuarta generación (T4G), abarca un amplio espectro de herramientas software, las cuales tienen la finalidad de facilitar al desarrollador de

software, la generación automática de código fuente basandose unicamente en la especificación del técnico.

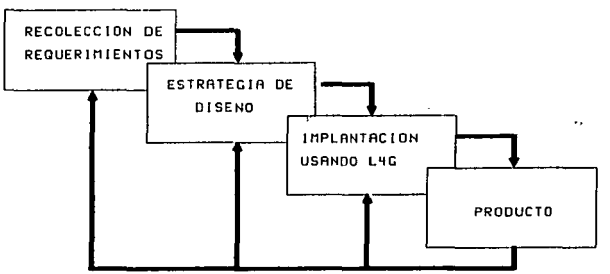


Fig. I.3. Técnicas de la cuarta generación.

Actualmente, un entorno para el desarrollo del software que soporte el paradigma T4G incluye algunas de las siguientes herramientas: lenguajes no procedimentales para consulta de bases de datos, generación de informes, manipulación de datos, interacción y definición de pantallas y generación de código; capacidades gráficas de alto nivel; y capacidad de hoja de cálculo.

El paradigma T4G para la ingeniería de programación se presenta en la Fig. I.3. Como se observa las T4G comienzan con el paso de recolección de requerimientos. Para aplicaciones pequeñas, puede ser posible ir directamente desde el paso de establecimiento

de los requerimientos a la implementación, usando un lenguaje de cuarta generación no procedimental (L4G). Sin embargo, es necesario un mayor esfuerzo para desarrollar una estrategia de diseño para el sistema, incluso si se utiliza un L4G. El uso de T4G sin diseño para grandes proyectos, causará las mismas dificultades (poca calidad, pobre mantenimiento, mala aceptación por el cliente) que se encuentra cuando se desarrolla software usando métodos convencionales.

El último paso de la Fig I.3. es el "producto". Para transformar una implementación T4G en un producto, se debe realizar una prueba completa, desarrollar una documentación con sentido y ejecutar todas las "actividades de transición" requeridas en los otros paradigmas de la ingeniería de programación.

CAPITULO II

CONCEPTO Y ANALISIS DEL SISTEMA

INTRODUCCION.

El desarrollo de sistemas puede estructurarse en forma general mediante dos componentes principales : el análisis y diseño de sistemas. Estos conceptos se refieren al proceso de examinar una situación de la empresa con la intención de mejorarla mediante nuevos procedimientos y métodos.

Las diferencias principales entre uno y otro es que el diseño de sistemas es el proceso donde se define la estructura así como la metodología de su funcionamiento, por lo tanto el análisis es el proceso de determinar la factibilidad del proyecto así como el de clarificar las necesidades del usuario.

II.1.- DEFINICION DE SISTEMA.

Existen varias definiciones de sistemas, pero en este caso, la que nos parece más completa y en términos más sencillos es la que define James A. Senn como:

" Un sistema es un conjunto de componentes que interactúan entre si para un cierto objetivo. Es una colección de elementos o medios que están relacionados y que pueden ser descritos en términos de sus atributos o de sus partes componentes ".

Los sistemas pueden ser abstractos o físicos. Un sistema abstracto es sólo conceptual, un producto de la mente humana. Esto es, no se puede ver o señalar como una entidad existente. Los sistemas sociales, religiosos y culturales son abstractos. Ninguna de estas entidades puede ser fotografiada, dibujada o representada gráficamente o de cualquier otra manera. Sin embargo, de hecho existen, y pueden ser discutidos, estudiados y analizados.

Un sistema físico, por lo contrario, es un conjunto de elementos materiales, que opera en relación con otro para lograr un objetivo o alcanzar una meta común. Un ejemplo de un sistema físico es:

-Un Sistema de Computación : Agrupamiento o conjunto de elementos de equipo (hardware) que trabajan interrelacionados bajo ciertos medios de control, con el objeto de procesar datos y producir informes de resultados.

El hecho de que un sistema pueda ser clasificado como abstracto o físico no implica ningún juicio de apreciación o valor. Esto es, un sistema físico no tiene mayor significado o es más necesario que un sistema abstracto. La diferencia es únicamente para fines analíticos.

II.1.1.- Elementos de un Sistema.

Un sistema es un conjunto de componentes que interaccionan entre sí para lograr un fin o propósito. Dentro de este marco básico de definición, pueden identificarse los elementos necesarios para la existencia de cualquier sistema. Estos elementos de sistema incluyen al entorno o medio circundante, límites o fronteras, entradas/salidas y componentes.

Entorno o medio circundante de un sistema: Todos los sistemas operan dentro de un entorno. Este es el medio ambiente que rodea al sistema, afectándolo y siendo afectado por él. Lo que denominemos entorno, depende de los objetivos del sistema, de sus necesidades y actividades, así como de si es físico o abstracto. Por ejemplo, un especialista en computación que estudie la eficiencia de la unidad procesadora central podría decir que el usuario es parte del entorno, ya que no forma parte de dicha unidad central. Sin embargo, si un especialista en computación está examinando el funcionamiento de un sistema de proceso por computadora de una organización, en términos de reducir costos de operación y minimizar errores en el manejo de los datos y de información, el usuario sería definitivamente parte del sistema. Estos ejemplos muestran de qué manera los diferentes ambientes pueden ser definidos para los sistemas. Además, el entorno es una función del individuo que observa o interactúa con el sistema.

Límites o fronteras de un sistema: Los límites o fronteras de un sistema demarcan o separan el entorno respecto del sistema. Este existe dentro de sus límites y todo lo que esté fuera de ellos constituye el ambiente. La línea que define los límites del sistema determina qué está incluido dentro del sistema y qué no lo está. Las características particulares de una frontera varían en función de que un sistema sea físico o abstracto. En un sistema físico el límite es una demarcación natural determinada por la estructura básica del sistema y por los objetivos y fines del mismo. En los sistemas abstractos, por otra parte, los límites son definidos típicamente por un observador.

La línea divisoria arbitraria puede, por lo tanto, variar de un observador a otro, a menos de que todos estén de acuerdo con los criterios para la selección. En cualquier circunstancia los límites de un sistema abstracto son determinados por el nivel de percepción del observador, la intención y el objetivo al determinar la línea y las nociones del observador, acerca del funcionamiento interno del sistema. En un sistema físico, los límites son fijos, como en el caso de la unidad central de procesamiento de una computadora, la cual contiene la unidad aritmético-lógica, la unidad de control y la unidad de almacenamiento primario. Sin embargo, las fronteras de un sistema abstracto, como en un grupo de trabajo de producción, pueden incluir o no al encargado del grupo, dependiendo de la definición del observador. Este último puede creer que el supervisor es parte del grupo de trabajo, pero otro creerá que el encargado queda fuera de los límites del grupo.

Entradas y salidas: El sistema interacciona con su ambiente por medio de los elementos de entrada y salida. Una entrada es cualquier cosa que ingresa al sistema proveniente del entorno; una salida es cualquier cosa que egresa del sistema, cruzando los límites hacia el medio circundante. En un sistema de cómputo, los datos ingresan al sistema como entrada y egresan del mismo como salida bajo la forma de información y de resultados de proceso. Los límites controlan cuidadosamente la entrada y la salida, regulando el flujo desde y hacia el sistema, y protegiéndolo de agentes destructivos o perjudiciales existentes en el ambiente. En esencia, los límites son los filtros de las entradas y salidas (fig. II.1).

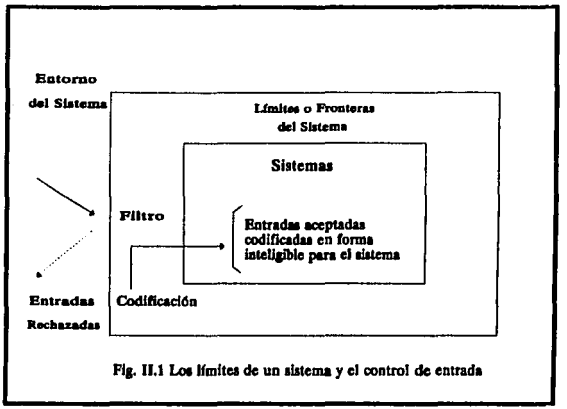


Fig. II.1 Los límites de un sistema y el control de entrada

Lo que inicialmente se pone en contacto con el sistema como entrada, a menudo no es lo que en realidad ingresa. Además de ser filtrado por los límites, la entrada es codificada ahí. Sin la codificación, gran parte de la entrada sería de poco o de ningún valor después de haber pasado los límites. Existen dos tipos de entradas: activantes y de mantenimiento. Las entradas activantes son los datos o la información sobre los que puede actuar el sistema para producir una salida. Por ejemplo un archivo que contiene transacciones de las cuentas por pagar, activa al sistema para actualizar los archivos y emite como salida informes de contabilidad. (fig. II.2).

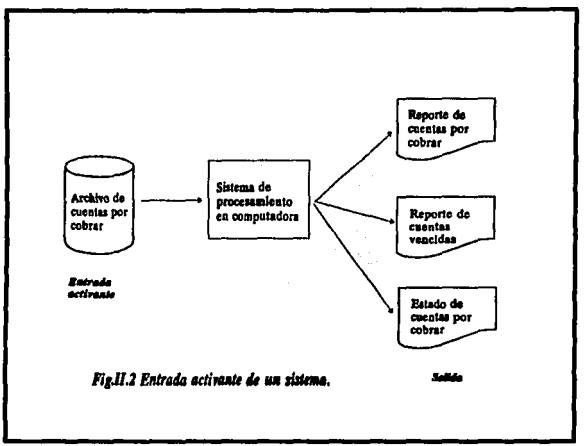


Fig.II.2 Entrada activante de un sistema.

Las entradas de mantenimiento están integradas estrechamente al sistema de control. Cuando se produce una salida, los datos frecuentemente son reunidos por los empleados de la empresa para determinar su adecuación y aceptación por el medio circundante. Estos datos son devueltos después al sistema como se muestra en la fig.II.3, y utilizados para regular o mejorar las actividades y los procesos del mismo.

Hasta ahora hemos descrito un sistema abierto, un sistema que intercambia información, materiales y energía con su entorno. Una característica adicional de un sistema abierto es la capacidad de adaptación, esto es, de ajustarse a los cambios en el medio circundante con miras a preservar su existencia. Los ajustes se efectúan tomando

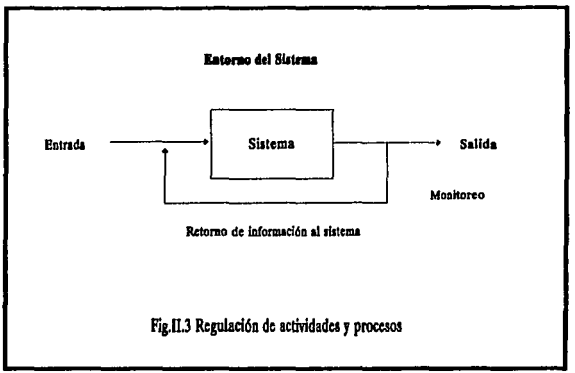


Fig.II.3 Regulación de actividades y procesos

como base las entradas y la retroalimentación (por ejemplo, las entradas que sirven para regular las actividades de un sistema de acuerdo con su objetivo o finalidad) a fin de que el sistema funcione en estado de equilibrio. En esencia, se tiene un proceso de ajuste dinámico debido a que las modificaciones se efectúan continuamente para asegurar que el sistema se encuentre aproximadamente en equilibrio. En caso de que cesará el proceso de ajuste, el sistema podría arruinarse o quedar completamente desorganizado. El término que se utiliza para designar la ruina o el deterioro de un sistema es la entropía. El proceso de recibir entradas del entorno para evitar el peligro de ruina o desgaste se llama entropía negativa. En contraste, un sistema cerrado esta autocontenido porque no interactúa con el medio circundante. La tendencia en un sistema cerrado es hacia la entropía, ya que no hay entradas desde el entorno para fomentar la adaptación. En otras palabras, sin entradas de mantenimiento no hay ajustes en un sistema cerrado, y la tendencia del sistema es hacia la ruina o el deterioro.

Componentes del sistema. Dentro de los límites o fronteras se encuentra todo el sistema, que puede ser una sola entidad o estar constituido por muchos componentes. Cuando un componente de sistema es en sí mismo un sistema, se le llama subsistema. Un componente de sistema puede ser definido como una unidad que trabaja con otros componentes (subsistemas) para lograr un fin específico, normalmente producir una salida (que puede ser la entrada de otra parte del sistema o a otro sistema completamente diferente, como se puede ver en la fig.II. 4). Las operaciones de cada componente separan, combinan o modifican de algún modo las entradas para cambiar su identidad y originar una salida. En el sistema debe haber correlación entre los subsistemas o componentes.

Esto es, tiene que existir un medio de transferencia de información entre los componentes, de manera que cada uno pueda realizar su tarea. La información es transferida o coordinada entre los componentes o subsistemas a través de interfaces, los elementos de interconexión (o intermediarios) en los límites de un sistema (o subsis-

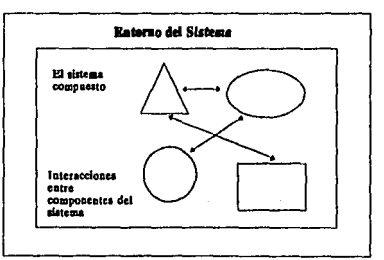


Fig.II.4 Un sistema compuesto de subsistemas

tema) que dan paso a información a través de ellos. Una interfaz codifica o decodifica información (o bien, energía) a una forma que el sistema pueda utilizar. Por ejemplo, en un sistema de cómputo, un canal de datos sirve como interfaz entre la unidad central de procesamiento y los dispositivos de entrada

II.2.- Análisis del Sistema.

El análisis de sistemas es la investigación y determinación de deseos, necesidades, objetivos y políticas del usuario en la realización de un sistema.

Tal vez la consideración más importante es que el analista no es el usuario. Hay que considerar siempre la manera de pensar y las características propias del usuario, así como su manera de trabajar y siempre que sea posible se debe buscar una interacción. El analista no sabe más de los procedimientos que el usuario (aunque en la mayoría de las veces el analista termina dominando más los procedimientos que el propio usuario).

Hay que hacer del usuario un amigo o el sistema fracasará por muy bueno que sea. Generalmente se espera que el analista estudie y diseñe sistemas de óptimo comportamiento. Esto es, hay que producir al mínimo costo posible una salida que cumpla completamente con los objetivos del usuario.

El profesional que forma parte de un grupo que analiza sistemas, debe tener los conocimientos de su especialidad que le permitan estudiar determinados aspectos particulares de un sistema; y contar, además, con aquellos conocimientos operacionales de diversas disciplinas distintos a los de su campo particular de actuación a fin de que puedan integrarse a un grupo de trabajo interdisciplinario y comunicarse con el resto del mismo. Esta comunicación resulta indispensable para los integrantes del grupo, pues con ella es factible dar al problema una solución que contenga todos los factores relevantes.

Aunque las actividades del analista son muy variadas, se podrían resumir en la siguiente lista sus principales obligaciones:

- Ofrecer soporte a la administración del centro de cómputo, sobre la planeación de nuevos sistemas y aplicaciones, y su organización, operación y control.
- Estudiar las áreas problema y hacer recomendaciones para mejorarlas.
- Evaluar los sistemas actuales y buscar las áreas en las que se puedan mejorar.
- Diseñar nuevos sistemas o rediseñar sistemas viejos y preparar las especificaciones de los sistemas al detalle.
- Analizar los requerimientos físicos y hacer las recomendaciones pertinentes.
- Preparar la documentación, desarrollar estándares y elaborar los reportes correspondientes en su área de procesamiento de información.
- Mantenerse actualizado en nuevas técnicas, que puedan aplicarse al desarrollo y mejoramiento de los sistemas.
- Debe ser capaz de definir las actividades de los programadores, operadores y otros trabajadores con los que tenga que tratar, así como entender el punto de vista de los administradores, líderes de proyecto y sobre todo de los usuarios.

La mejor manera de introducir un sistema altamente automatizado en una organización es reduciendo al mínimo los problemas psicológicos o de ajustes de personal durante todo el proceso, especialmente durante el análisis y el diseño. Esto implica un constante intercambio de ideas y sugerencias entre los analistas, los usuarios, los administradores y los trabajadores afectados por el sistema.

A continuación se mencionan los principales pasos para el análisis de un sistema:

- Definición del problema.
- Determinación de los objetivos del sistema.
- Organización para el estudio del sistema, equipos de trabajo.
- Recopilación de información.
- Estudio de otros sistemas.
- Entrevistas.
- Análisis de documentos.
- Lista de recursos disponibles.
- Análisis requeridos.
- Manejo de transacciones.
- Reportes de referencia.
- Muestreos y estadísticas.
- Observación del personal.
- Diagrama de bloques.
- Alternativas de operación.
- Presentación del estudio para aprobación.
- Aprobación del estudio.

II.3.- ETAPAS DEL ANALISIS.

Como se mencionó anteriormente el análisis de sistemas es una técnica o metodología, y como tal esta compuesta de las siguientes etapas:

II.3.1.- Diagnóstico de la situación actual.

Se puede iniciar una petición por muchas razones, pero la clave es que alguien, ya sea un gerente, un empleado o un especialista de sistemas, inicie un requerimiento para recibir ayuda de un sistema de programación. Cuando ese requerimiento se realiza, la primera actividad de sistemas, es decir, la investigación preliminar, se inicia. Esta actividad tiene tres partes: clarificación del requerimiento, estudio de factibilidad y aprobación del requerimiento. El resultado será aprobar el requerimiento para una atención posterior o rechazarlo como no factible para un desarrollo futuro.

II.3.2.- Clarificación del requerimiento.

En las empresas muchos requerimientos de los empleados y usuarios no están establecidos claramente; por lo tanto, antes de que pueda considerarse la investigación del sistema, el proyecto requerido debe examinarse para determinar con precisión lo que desea la empresa.

Por otro lado, la persona que hace el requerimiento puede estar simplemente pidiendo ayuda sin saber qué es lo que está mal o por qué existe un problema. La clarificación del problema en este caso es mucho más difícil; en cualquier caso, antes de poder llegar a otro paso, el requerimiento de proyecto debe estar claramente establecido.

II.3.3.- Análisis de Factibilidad.

Todos los proyectos son realizables, dados recursos ilimitados y tiempo infinito. Desafortunadamente, el desarrollo de un sistema basado en computadora se caracteriza por la escasez de recursos y la dificultad (si no imposibilidad) de los plazos de entrega. Es necesario y prudente evaluar la factibilidad de un proyecto lo antes posible. Se pueden evitar meses o años de esfuerzo, miles de millones de dólares y una inversión profesional incontable si un sistema mal concebido es reconocido al principio de la etapa de definición.

Se estudian tres pruebas de factibilidad: Operativa, Técnica y Financiera; todas con la misma importancia.

II.3.3.1.- Factibilidad Operativa.

Los proyectos propuestos son benéficos sólo si pueden convertirse en sistemas que cumplan los requerimientos operativos de la compañía. Dicho sencillamente, esta prueba de factibilidad cuestiona si el sistema trabajará cuando se instale y desarrolle. A continuación se presentan algunas preguntas que ayudarán a probar la factibilidad operativa de un proyecto.

- ¿Existe suficiente apoyo para el proyecto por parte de la gerencia? ¿También de los usuarios? Si el sistema actual gusta y se usa, al grado de que las personas no ven ninguna razón para cambiarlo, ¿Puede haber resistencia?

- ¿Son aceptables los métodos actuales del negocio para los usuarios? Si no lo son, los usuarios pueden aceptar un cambio que permita tener un sistema más operativo y útil.

- ¿Se han involucrado los usuarios en la planeación y desarrollo del proyecto? Una participación al iniciar el proyecto reduce las posibilidades de resistencia al sistema y al cambio e incrementa la posibilidad de proyectos exitosos.

- ¿Causará daño el sistema propuesto?
- ¿Producirá resultados más pobres en algún aspecto o área?
- ¿Dará como resultado una pérdida de control en alguna área?
- ¿Se perderá el acceso a la información?
- ¿Será más pobre que antes el desempeño individual después de la puesta en marcha?
- ¿Se afectará a los clientes en forma indeseable?
- ¿Disminuirá la rapidez del trabajo en algunas áreas?

Los aspectos que son relativamente pequeños y parecen cuestiones de menor importancia al principio, encuentran siempre maneras de crecer y convertirse en problemas mayores después de la puesta en marcha; por lo tanto, todos los aspectos operativos deben considerarse con cuidado.

II.3.3.2.- Factibilidad Técnica.

Los aspectos técnicos que normalmente surgen durante la etapa de factibilidad de la investigación son:

- ¿Existe la tecnología necesaria (o puede adquirirse) para hacer lo que se sugiere?
- ¿Tiene el equipo propuesto la capacidad técnica para almacenar los datos requeridos y utilizarlos en el nuevo sistema?
- ¿El sistema propuesto y sus componentes proporcionarán respuestas adecuadas a las preguntas, sin importar el número o ubicación de los usuarios?
- ¿Se puede ampliar el sistema si se desarrolla?

- ¿Existen garantías técnicas de exactitud, confiabilidad, facilidad de acceso y seguridad en los datos?

II.3.3.3.- Factibilidad Financiera y Económica.

Un sistema que puede desarrollarse técnicamente y que se utilizará si se instala, debe considerarse como una buena inversión para la empresa, es decir, los beneficios financieros deben igualar o exceder los costos financieros. Las preguntas económicas y financieras que se plantean los analistas durante la investigación preliminar buscan estimaciones de:

- El costo de llevar a cabo una investigación completa de sistemas.
- El costo del hardware y software para el tipo de aplicación considerado.
- El costo si nada cambia (si el sistema no se desarrolla).

Si queremos hacer un análisis más detallado podemos dividir aún más los costos:

COSTOS FIJOS si no cambian con el volumen de producción, como salarios, renta de equipo (a menos que se rente tiempo), local, aire acondicionado y otros similares que no varían, se use o no se use la computadora.

COSTOS VARIABLES son aquellos que cambian en función del volumen de producción, como papelería, tarjetas, etc.

Por otro lado podemos hacer otras divisiones:

COSTOS CONTROLABLES si se pueden controlar por medio de métodos y procedimientos. Son los que el analista debe estudiar y cuidar con más detenimiento, por ejemplo, el tiempo de máquina, costos de suministro, etc.

COSTOS NO CONTROLABLES son aquellos fuera del control de la organización, como impuestos, licencias y otras tarifas gubernamentales. Simplemente hay que reportarlos exactamente.

Otra división puede ser, desde el punto de vista de frecuencia:

COSTOS UNICOS si sólo ocurre una vez durante la implantación del sistema, como el costo de la fase de análisis, la del diseño, la de la programación, la de la instalación, etc.

COSTOS REPETITIVOS son aquellos que ocurren periódicamente y son necesarios para mantener al sistema operando. Entre éstos están salarios, suministros, renta de equipo, etc.

Los costos para sistemas de cómputo, los podemos dividir en únicos y repetitivos, entre los primeros tenemos los siguientes:

- *La Transición del Sistema Viejo al Nuevo.*- Tales costos, como el contratar tiempo de máquina, o una computadora, gastos de transporte e instalación, personal adicional o tiempos extra y otros similares, entran en este renglón.

- *Costos de Conversión.*- La conversión de archivos y datos para la computadora, costos de preparación de la información, capacitación del personal, etc.

- *Costos de Análisis y Diseño.*- Los costos de todo el personal involucrado en éstas fases.

- *Costos de Programación.*- Todos los costos incurridos durante el desarrollo y prueba de los programas del sistema.

- *Consultores y Asesores.*- Si es necesario requerir de este tipo de servicios; generalmente sólo durante la etapa de implantación del sistema.

- *Tiempo de Máquina.*- Hay que hacer notar la diferencia entre el tiempo de máquina usado durante la implantación del sistema en pruebas y desarrollos, y el tiempo consumido por la operación normal del sistema.

Los Costos Repetitivos principales son:

- *Preparación de Datos.-* Codificación, perforación y/o captura de las transacciones del sistema, incluyendo las verificaciones necesarias para su confiable utilización dentro del sistema.

- *Personal.-* Los costos del personal involucrado en la operación del sistema, algunos de ellos sólo son parte de su tiempo en la operación del sistema, como los operadores, las capturistas y a veces los programadores.

- *Renta o Depreciación del Equipo.-* Este renglón es bastante complejo para el caso de instalaciones que soportan multiprogramación y tiempo compartido, ya que hay que dividir el uso de los recursos de una manera justa entre los diferentes sistemas en operación.

- *Energía y Control Ambiental.-* Los costos de energía eléctrica y de depreciación de plantas de emergencia, fuentes ininterrumpibles de energía, aire acondicionado y sistemas de seguridad se deben prorratear para cada sistema, sobre bases similares al punto anterior.

- *Suministros.-* Formas continuas, papelería y cintas de impresora se pueden considerar como gastos, pero en general los medios magnéticos se deprecian.

- *Mantenimiento de Programación.-* Siempre es necesario hacer cambios menores al sistema, por lo que en general se asigna personal de programación para el soporte de mantenimiento.

- *Mantenimiento del equipo.-* Generalmente el proveedor establece pólizas de mantenimiento o servicio por llamada.

- *Capacitación.-* Los cambios al sistema y la rotación de personal, frecuentemente originan gastos repetitivos de capacitación.

- *Seguros.-* Siempre es conveniente tener asegurado un sistema de cómputo.

- *Renta de Local.-* Aún cuando el espacio ocupado por el sistema de cómputo sea propiedad de la misma organización, tiene un costo y hay que considerarlo.

- *Comunicaciones.-* Si se trabaja en un ambiente de teleproceso, existirán costos como mantenimiento de líneas o renta de líneas telefónicas, permisos y equipo de telecomunicaciones.

- *Transporte.-* En algunos casos la información se envía de su fuente de origen al centro de proceso mediante autobús, correo, avión, etc.

Básicamente la factibilidad económica consiste en determinar si un nuevo sistema es más económico que el anterior (en caso de haber ya un sistema), considerando todas las característica (técnicas o no) de ambos sistemas. Para ésto, hay que obtener los costos anteriormente delineados para ambos sistemas y compararlos..

II.3.3.4.- Manejo de proyectos no factibles.

No todos los proyectos propuestos para su evaluación y revisión se juzgan como aceptables. ¿Qué les sucede?

Las solicitudes que no pasan las pruebas de factibilidad no continúan más adelante, a menos de que quiénes las originaron trabajen en ellas y las vuelvan a someter como nuevas propuestas. En algunos casos, sólo parte del proyecto es realmente inoperante. Aquí, el comité de selección puede decidir combinar la parte operante del proyecto con otra propuesta factible. En otros casos, el comité puede regresar el proyecto con la sugerencia de que se modifique ciertos procedimientos manuales, utilizados en el departamento, para mejorar las operaciones.

II.3.4.- Aprobación de los requerimientos.

No todos los proyectos requeridos son deseables o factibles. (De hecho, algunas compañías reciben tantas peticiones de proyectos de los empleados que solamente se llevan a cabo unas cuantas). Sin embargo, aquellos que son tan factibles como deseables deben anotarse para tomarlos en cuenta.

En algunos casos, el desarrollo puede comenzar inmediatamente, pero en la mayor parte, los miembros del departamento de sistemas están ocupados en otros proyectos que se encuentran en marcha. Cuando esto sucede, la gerencia decide qué proyectos son más importantes y entonces los programa. Después de que se aprueba la requisición de un proyecto, se estima su costo, la prioridad, el tiempo de terminación y los requerimientos de personal, para determinar en qué lista de proyectos se incluirá.

CAPITULO III.

PLANEACION DEL SISTEMA DE PROGRAMACION

INTRODUCCION.

Para llevar a cabo un buen proyecto de desarrollo de software, debemos comprender el ámbito del trabajo a realizar, los recursos requeridos, las tareas a ejecutar, las referencias a tener en cuenta, el esfuerzo (costo) a emplear y la agenda a seguir.

La planificación del proyecto de software combina dos tareas: investigación y estimación. La investigación nos permite definir el alcance del elemento software de un sistema informático. El segundo paso de la planificación del software es la estimación, y por tanto, una característica de la estimación es la incertidumbre.

La estimación de recursos, de costos, y de agendas para el esfuerzo de desarrollo de software requiere experiencia, acceso a una buena información histórica, y coraje para confiar en medidas cuantitativas cuando todo lo que existe son datos cualitativos. La estimación lleva un riesgo ya que la complejidad del proyecto tiene un gran efecto en la

incertidumbre inherente de la planificación. La complejidad sin embargo, es una medida relativa que se ve afectada por la familiaridad con anteriores esfuerzos.

El riesgo en la planificación se mide por el grado de incertidumbre en las estimaciones cuantitativas establecidas para los recursos, costos y métodos, y de esta forma el planificador y, más importante aún, el cliente, debe reconocer que cualquier cambio en los requerimientos del software, significa inestabilidad en el costo y en el método.

El objetivo del planificador del proyecto de software es el suministrar un área de trabajo que permita al director hacer estimaciones razonables de recursos, costos y métodos. Estas estimaciones se hacen, sin un marco de tiempo limitado, al principio de un proyecto de software y deben ser actualizadas regularmente al progresar el proyecto.

III.1. EL ALCANCE DEL SISTEMA DE PROGRAMACION

La primera actividad en la planificación de un proyecto de software es determinar el alcance del software. La función y el rendimiento asignados al sistema de computadora deben ser valorados para establecer un alcance del proyecto que no sea ambiguo ni incomprensible a nivel de directivos y técnicos. La Fig. III.1. muestra los tópicos que se presentan como parte del alcance del sistema de programación.

La declaración del alcance del software debe estar delimitada, por lo tanto, los datos cuantitativos (por ejemplo, número de usuarios simultáneos, tiempo máximo de respuesta posible) estarán explícitamente establecidos; las restricciones y/o limitaciones, han de ser señaladas; y los factores de mitigación (por ejemplo, los algoritmos deseados sean bien entendidos) han de ser descritos.

Las funciones del software serán evaluadas y en algunos casos refinadas para suministrar más detalles. Las consideraciones de desarrollo abarcan las restricciones de tiempo de

procesamiento, las limitaciones de la memoria para el software, y las características especiales dependientes de la máquina.

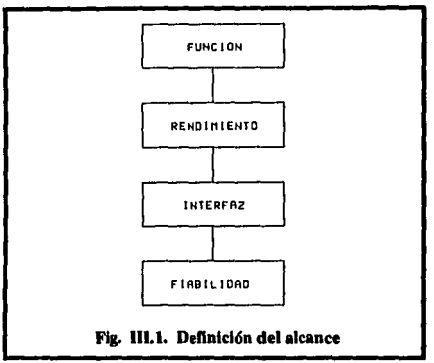


Fig. III.1. Definición del alcance

El planificador debe considerar la naturaleza y complejidad de cada interfaz para determinar cualquier efecto en los recursos, costos y métodos del desarrollo. El concepto de interfaz abarca lo siguiente:

- Hardware (procesador y periféricos) que ejecutan el software y dispositivos (máquinas y pantallas) controlados indirectamente por el software.
- Software ya existente (como son: rutinas de acceso a base de datos, paquetes de subrutinas y sistemas operativos) y que debe ser instalado con el nuevo software.
- Personal que hace uso del software por medio de terminales u otros medios de entrada/salida.
- Procedimientos que preceden o suceden al software como una serie secuencial de operaciones.

El aspecto menos preciso del alcance del software es una discusión de su fiabilidad, las medidas de fiabilidad del software se encuentra en la etapa de formación del desarrollo. Las características de fiabilidad del hardware, tales como el tiempo medio entre fallos (TMEF) son difíciles de trasladar al dominio del software. Sin embargo, la naturaleza general del software puede dictar consideraciones especiales para asegurar la confiabilidad. Por ejemplo un sistema de control de inventario o un procesador de textos no debería de fallar, pero el impacto del fallo es considerablemente menos dramático, que si fallara un sistema de control de tráfico aéreo o de transporte espacial (ambos sistemas con valor humano).

Aunque puede no ser posible cuantificar la fiabilidad del software de forma tan precisa como quisiéramos, podemos utilizar la naturaleza del proyecto como ayuda para la formulación de estimaciones de esfuerzo y costo que aseguren la fiabilidad.

III.2. RECURSOS

La segunda tarea de la planificación del desarrollo de software es la estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software. La Fig. III.2. muestra los recursos de desarrollo como una pirámide. En la base, deben existir las herramientas hardware y software que soporten el desarrollo de software. En el nivel más alto, siempre se requiere el recurso primario la gente.

Cada recurso se especifica con cuatro características: descripción del recurso; informe de disponibilidad; fecha cronológica en la que se requiere el recurso; y tiempo en que será aplicado el recurso.

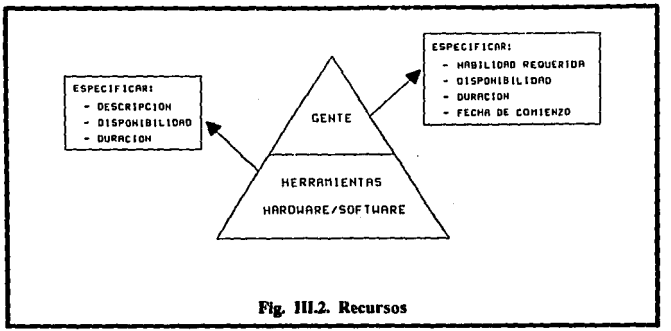


Fig. III.2. Recursos

III.2.1. Recursos Humanos

Entre los principales problemas planteados por la crisis del software, ninguno es más inquietante que la relativa escasez de recursos humanos capaces para la elaboración de software, ya que la gente es el requerimiento primario para su obtención.

El planificador comienza evaluando el alcance y seleccionando las técnicas requeridas para el desarrollo, también se ha de especificar tanto la posición de la organización (director, ingeniero en programación, etc.) como la especialidad (telecomunicaciones, bases de datos, microprocesadores, etc.).

Para proyectos relativamente pequeños, una sola persona (persona-año) puede llevar a cabo todas las etapas del software, consultando con especialistas siempre que lo requiera. Para proyectos amplios, la participación varía a lo largo del ciclo de vida.

La Fig. III.3. ilustra en un proyecto típico la participación para cada etapa de la ingeniería en programación.

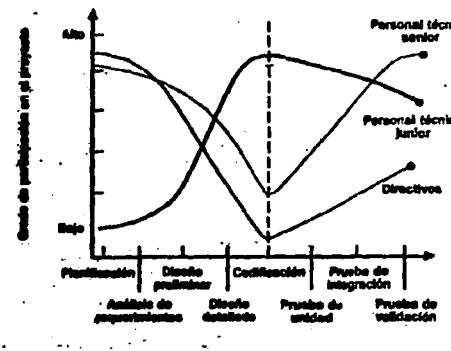


Fig. III.3. Grado de participación en un proyecto.

Haciendo referencia a la figura, la participación de la dirección se da al principio del ciclo de vida, disminuye a niveles más bajos durante las etapas centrales de la fase de desarrollo y crece a medida que se acerca la conclusión del proyecto. El personal técnico senior también participa activamente durante la planificación del proyecto, el análisis de los requerimientos, el diseño, y las etapas de pruebas finales. El personal técnico junior esta más involucrado durante las últimas etapas del diseño, en la codificación y en la etapa de pruebas iniciales. El número de personal requerido para un proyecto de software, sólo

puede ser determinado después de hacer una estimación del esfuerzo del desarrollo (por ejemplo, personas-mes o personas-año).

III.2.2. Recursos Hardware

Dentro del contexto de recursos, el hardware también puede ser una herramienta para el desarrollo de software. Se deben considerar tres categorías de hardware durante la planificación del proyecto de software: el sistema de desarrollo, la máquina objetivo y otros elementos de hardware del nuevo sistema.

El sistema de desarrollo (también llamado sistema anfitrión) consiste en la computadora y los periféricos asociados que se utilizarán durante las etapas de desarrollo del software. Se utiliza el sistema de desarrollo porque puede soportar múltiples usuarios, mantener amplios volúmenes de información que pueden ser compartidos por los miembros del equipo de desarrollo de software y soportar una gran variedad de herramientas de software. Excepto para grandes proyectos, el sistema de desarrollo no tiene por que ser especialmente adquirido. Así el recurso hardware puede ser visto más como un acceso a una computadora existente que como la adquisición de una nueva computadora.

La máquina objetivo es un procesador que ejecuta software como parte del sistema basado en computadora. En la mayoría de las aplicaciones de minicomputadoras y de grandes sistemas, la máquina objetivo y el sistema de desarrollo son idénticos.

Se pueden especificar como recursos para el desarrollo de software otros elementos de hardware del sistema basado en computadora. Por ejemplo, el software de control numérico (CN) utilizado en ciertas máquinas como parte de la etapa de validación, o un proyecto de software de composición automática puede necesitar un fotocompositor en algún momento de desarrollo.

III.2.3. Recursos Software

Al igual que utilizamos el hardware como una herramienta para construir nuevo hardware, utilizamos el software para ayudar al desarrollo del nuevo software. La primera aplicación del software para el desarrollo de software fue la reconstrucción. Se escribió un primitivo traductor de lenguaje ensamblador en lenguaje máquina y se uso para desarrollar un ensamblador más sofisticado.

Hoy en día existen tres extensas categorías de herramientas de software que están disponibles para el ingeniero en programación:

Herramientas orientadas al código. Las herramientas de esta categoría son a menudo las únicas herramientas disponibles para el desarrollo de software. Las herramientas orientadas al código incluyen: compiladores de lenguajes de programación, editores, enlazadores y cargadores; ayuda para la depuración; y una amplia serie de utilidades específicas del lenguaje.

Herramientas de metodología. De la misma forma que un constructor necesita de una pala para excavar los cimientos, antes de utilizar un martillo y una sierra para construir una casa, el ingeniero en programación necesita herramientas de metodología para analizar y diseñar, antes de codificar. Las herramientas de esta categoría dan soporte a la planificación del proyecto, al análisis de los requerimientos, al diseño, a las pruebas, a la gestión de configuraciones, al mantenimiento y a otras actividades.

Herramientas de cuarta generación. Para aplicar las técnicas de la cuarta generación (T4G), tiene que estar disponible un conjunto de herramientas especiales que permitan al equipo de desarrollo de software especificar problemas utilizando lenguajes de acceso a bases de datos y otras técnicas no procedimentales. Entre la muchas herramientas disponibles de esta categoría están NOMAD, FOCUS, INTELLECT (lenguajes de

petición de base de datos), y GAMMA, HOS, TELON, TRANSFORM (generadores de código).

La investigación en herramientas del software y en el desarrollo de entornos de programación van de la mano. los entornos avanzados de la ingeniería en programación pueden utilizar técnicas de sistemas expertos para ayudar al analista y diseñador en el desarrollo de sistemas informáticos.

Cualquier discusión de recursos del software quedaría incompleta sin considerar la reusabilidad, o sea, la creación y reutilización de los bloques de software. Dichos bloques deben ser catalogados para su fácil referencia, estandarizados para una fácil aplicación y validados para su fácil integración.

Cuando el software reutilizable está especificado como un recurso, dos "reglas" debe considerar el planificador del software:

- Si el software existente satisface los requerimientos, adquirirlo. El costo del software adquirido será casi siempre menor que el costo de desarrollo de software equivalente.

- Si el software existente requiere alguna modificación antes de que pueda ser propiamente integrado en el sistema, proceder con cuidado. El costo de modificar software existente puede a veces ser mayor que el costo de desarrollo de software equivalente.

La mayoría de los observadores de la industria están de acuerdo en que la mejora de la productividad del desarrollo de software y de la calidad del producto acabarán con la crisis del mismo.

III.3. ESTIMACION DE COSTOS

En los inicios de la informática, el costo del software representaba un pequeño porcentaje del costo total del sistema informático, basado en computadora. Hoy en día, el software es el elemento más caro en muchos sistemas informáticos, y un error considerable en la estimación del costo puede marcar la diferencia entre beneficios y pérdidas, sobrepasar el costo puede ser desastroso para el equipo de desarrollo.

La estimación del costo y del esfuerzo en el desarrollo del software nunca será una ciencia exacta, debido a que intervienen demasiadas variables - variables humanas, técnicas, de entorno, políticas- las cuales pueden afectar el costo final del software y al esfuerzo aplicado para desarrollarlo. Sin embargo, la estimación del proyecto de software puede transformarse de un oscuro arte en una serie de etapas que proporcionen estimaciones con un grado de riesgo aceptable.

Para realizar estimaciones seguras de costo y esfuerzo surge un número de opciones posibles:

- Retrasar la estimación más adelante en el proyecto.
- Utilizar "técnicas de descomposición" relativamente simples para generar las estimaciones del proyecto de software.
- Desarrollar un modelo empírico para el costo y el esfuerzo del software.
- Adquirir una o más herramientas automáticas de estimación.

La primera opción, aunque atractiva, no es práctica: las estimaciones del costo deben ser proporcionadas "de antemano".

Las tres opciones restantes son aproximaciones viables para la estimación del proyecto de software. Las técnicas de descomposición utilizan una aproximación de "divide y vencerás" para la estimación del proyecto de software, los modelos de estimación empírica pueden utilizarse para complementar las técnicas de descomposición y ofrecer una aproximación de la estimación potencialmente evaluable por ella misma.

Las herramientas automáticas de estimación consideran una o más técnicas de descomposición o modelos empíricos, y cuando se combinan con una interfaz hombre-máquina, las herramientas automáticas proporcionan una atractiva opción para la estimación.

Cada una de las opciones viables de la estimación de costos del proyecto de software, es o será sólo tan buena como lo sean los datos históricos, la evaluación del costo se queda en un fondo muy inestable.

III.3.1. Técnicas de Descomposición

Los seres humanos han desarrollado una aproximación natural a la resolución de problemas: si el problema a resolver es demasiado complicado, tendemos a subdividirlo hasta encontrar problemas más manejables.

En la estimación del proyecto de software es una forma de resolución de problemas y, en la mayoría de los casos, el problema a resolver (o sea, desarrollar estimaciones de costo y esfuerzo para un proyecto de software) es demasiado complejo para considerarlo como una sola pieza. Por esta razón, descomponemos el problema, re-caracterizándolo como un conjunto de pequeños problemas esperando que sean más manejables para su solución.

III.3.1.1. Estimación LDC y PF

Las estimaciones LDC (Líneas de Código) y PF (Puntos de Función), son técnicas de estimación diferentes, pero ambas tienen características comunes. Los datos LDC y PF se utilizan de dos maneras durante la estimación del proyecto de software:

- Como una estimación variable que se utiliza para "tallar" cada elemento del software.
- Como métricas de línea base colectadas de proyectos anteriores y utilizadas en conjunción con las estimaciones variables para desarrollar proyecciones de costo y esfuerzo.

El planificador del proyecto comienza con un informe limitado del ámbito de alcance del software y, partiendo de este informe, intenta descomponer el software en pequeñas subfunciones que puedan ser estimadas individualmente, entonces se estima LDC o PF (la variable de estimación) para cada subfunción. Las métricas de productividad son entonces aplicadas para la estimación a la variable apropiada y se obtiene el costo y el esfuerzo para la subfunción. Las estimaciones de la subfunción se combinan para producir una estimación global para el proyecto completo.

Las técnicas de estimación LDC y PF difieren en el nivel de detalle requerido para la descomposición. Cuando se utiliza LDC como una variable de estimación, la función de descomposición es absolutamente esencial y se toma a menudo a través de considerables niveles de detalle. Dado que los datos requeridos para estimar los puntos de función son más macroscópicos, el nivel de descomposición utilizado, cuando PF es la variable de estimación, es considerablemente menos detallado. Debe también señalarse que LDC es determinada directamente, mientras que PF es determinada indirectamente por la

estimación del número de entradas, archivos de datos, preguntas, e interfaces externas. Utilizando datos históricos o la intuición, el planificador estima valores LDC o PF optimista, más probable, y pesimista para cada función. Cuando se especifica un rango de valores se proporciona una indicación implícita del grado de incertidumbre.

El valor esperado para la variable de estimación E, puede obtenerse como una media ponderada de las estimaciones LDC o PF optimista (a), más probable (m), y pesimista (b) de las estimaciones LDC o PF. Por ejemplo:

$$E = (a + 4m + b)/6$$

Asumimos que es muy poca la probabilidad de que el actual resultado de LDC o PF caiga fuera de los valores pesimistas y optimistas.

Una vez que ha sido determinado el valor esperado para la variable de estimación, se aplican los datos de productividad de LDC o PF, y el planificador puede aplicar una de dos diferentes aproximaciones.

1.- El valor total de la variable de estimación para todas las subfunciones puede multiplicarse por la métrica de productividad media correspondiente a la variable de estimación. Por ejemplo, si asumimos que en total se estiman 310 PF y que la productividad media de PF basada en anteriores proyectos es de 5,5 PF/PM, entonces el esfuerzo global para el proyecto es:

Esfuerzo(PM) = 310/5,5 = 56 personas-mes

2.- El valor de la variable de estimación para cada subfunción puede multiplicarse por el valor ajustado de productividad que esta basado en el nivel de complejidad percibido de la subfunción. Para funciones de complejidad media se utiliza la métrica de productividad media. Sin embargo, la métrica de productividad media se ajusta arriba o abajo, dependiendo de que sea mayor o menor que la complejidad media de una subfunción particular. Por ejemplo, si la productividad media es de 490 LDC/PM, las estimaciones de líneas de código para las subfunciones que son considerablemente más complejas que la media puede multiplicarse por 300 LDC/PM y las funciones simples por 650 LDC/PM.

La única respuesta posible razonable a la pregunta ¿Son correctas las estimaciones? es: "No podemos asegurarlo". Cualquier técnica de estimación, no importa cómo sea de sofisticada, tiene que ser comprobada nuevamente con otra aproximación.

III.3.1.2. Ejemplo:

Como ejemplo de técnicas de estimación LDC y PF, consideremos un paquete de software a desarrollar para una aplicación de diseño asistido por computadora (CAD). Revisando la especificación del sistema encontramos que el software va a ejecutarse en una estación de trabajo de microcomputadora y se conectará con varios periféricos gráficos incluyendo ratón, digitalizador, pantalla en color de alta resolución, y una impresora de alta resolución. Para los propósitos de este ejemplo, se utilizará LDC como variable de estimación.

La evaluación del alcance indica que se requieren las siguientes funciones principales para el software CAD:

- Interfaz de usuario y facilidades de control (IUCF)
- Análisis geométrico bidimensional (AG2D)

- Análisis geométrico tridimensional (A3GD)
- Gestión de estructuras de datos (GED)
- Facilidades de visualización de gráficos de computadora (FVGC)
- Control de periféricos (CP)
- Módulos de análisis de diseño (MAD)

Siguiendo la técnica de descomposición, se desarrolla la tabla de estimación, mostrada en la Fig. III.4. Se desarrolla un rango de estimaciones LDC. Revisando las tres primeras columnas de la tabla, puede verse que el planificador es bastante correcto en LDC requeridas para la función de control periférico (sólo 450 líneas de código separan las estimaciones optimistas de las pesimistas). Por otro lado, la función de análisis geométrico tridimensional es relativamente desconocida como indican las 4000 LDC de diferencia entre los valores optimistas y los pesimistas.

Los cálculos para el valor esperado se realizan para cada función y se colocan en la cuarta columna de la tabla (Fig. III.5.). Sumando verticalmente en la columna del valor esperado, se establece una estimación de 33,360 líneas de código para el sistema CAD.

El resto de la tabla de estimación requerido para la técnica de descomposición se muestra en la Fig. III.6. En este caso, el planificador utiliza diferentes valores de las métricas de productividad para cada función basándose en el grado de complejidad. Los valores contenidos en las columnas de costo y meses de la tabla se determinan por la multiplicación o división de LDC esperadas por $/LCD y LDC/personas-mes respectivamente. El total de estas dos columnas es de $67,000 y 145 personas-mes.

FUNCION	OPTIMISTA	MAS PROBABLE	PESIMISTA	ESPERADO	S/LINEA	LINEA/MES	COSTO	MESES
CONTROL DE LA INTERFAZ DE USUARIO	1900	2400	2850					
ANALISIS GEOMETRICO EN 2-D	4100	5200	7400					
ANALISIS GEOMETRICO EN 3-D	4500	6000	8500					
GESTION DE LA ESTRUCTURA DE DATOS	2850	3400	3600					
VISUALIZACION DE GRAFICOS EN COMPUTADOR	4850	4900	6200					
CONTROL PERIFERICO	2000	2100	2450					
ANALISIS DE DISEÑO	5600	8500	9800					

Fig. III.4. Tabla de estimación de esfuerzos

FUNCION	OPTIMISTA	MAS PROBABLE	PESIMISTA	ESPERADO	S/LINEA	LINEA/MES	COSTO	MESES
CONTROL DE LA INTERFAZ DE USUARIO	1900	2400	2850	2390				
ANALISIS GEOMETRICO EN 2-D	4100	5200	7400	5300				
ANALISIS GEOMETRICO EN 3-D	4500	6000	8500	6000				
GESTION DE LA ESTRUCTURA DE DATOS	2850	3400	3600	3250				
VISUALIZACION DE GRAFICOS EN COMPUTADOR	4850	4900	6200	4950				
CONTROL PERIFERICO	2000	2100	2450	2140				
ANALISIS DE DISEÑO	5600	8500	9800	8480				
TOTAL				33360				

Fig. III.5. Tabla de estimación de esfuerzos

FUNCION	OPTIMISTA	MAS PROBABLE	PESIMISTA	ESPERADO	$LINEA	LINEA/MES	COSTO	MESES
CONTROL DE LA INTERFAZ DE USUARIO	1800	2400	2650	2341	14	315	32.774	7.43
ANALISIS GEOMETRICO EN 2-D	4100	5200	7400	5383	20	220	107.660	24.47
ANALISIS GEOMETRICO EN 3-D	4500	6400	8150	6408	20	220	128.080	30.01
GESTION DE LA ESTRUCTURA DE DATOS	2850	3400	3600	3350	10	240	80.400	18.09
VISUALIZACION DE GRAFICOS EN COMPUTADORA	4050	4900	6240	4975	22	200	109.450	24.08
CONTROL PERIFERICO	2000	2100	2450	2141	20	140	59.948	15.29
ANALISIS DE DISEÑO	5600	6150	6600	6140	10	300	151.200	20.00
TOTAL				33338			8857.478	146.87

Fig. III.6. Tabla de estimación de esfuerzos

III.3.2. Estimación del esfuerzo

La estimación del esfuerzo es la técnica más común para calcular el costo de un proyecto de ingeniería en programación, se aplica un número de personas-día, personas-mes o personas-año a la solución de cada tarea del proyecto. Se asocia un costo a cada unidad de esfuerzo y se obtiene el costo estimado.

La estimación del esfuerzo comienza con una delineación de las funciones del software obtenidas del alcance del proyecto, para cada una de las funciones debe realizarse una serie de tareas de ingeniería en programación - análisis de requerimientos, diseño, construcción y pruebas -.

El planificador del proyecto estima el esfuerzo (por ejemplo personas-mes) a emplear en cada tarea de la ingeniería en programación y en cada función del software. Estos datos constituyen la matriz central de la tabla de la Fig. III.7. Se aplican los coeficientes de trabajo (o sea, costo/unidad de esfuerzo) a cada una de las tareas.

Como último paso se calculan los costos y el esfuerzo para cada una de las funciones y tareas de la ingeniería en programación. Si la estimación del esfuerzo es calculada independientemente de la estimación LDC y PF, tenemos ahora dos estimaciones de costo y esfuerzo que pueden compararse y ponerse en común. Si ambos conjuntos de estimaciones muestran concordancias razonables, habrá una buena razón para creer que las estimaciones son fiables. Si, por otro lado los resultados de estas técnicas de descomposición están poco de acuerdo, debe realizarse otra investigación y análisis.

III.3.2.1. Ejemplo:

Para ilustrar el uso de la estimación del esfuerzo, consideremos nuevamente el software de CAD. La configuración del sistema y todas las funciones del software permanecen iguales y son obtenidas del alcance del proyecto.

Refiriendonos a la tabla de estimación de esfuerzos mostrada en al Fig. III.8, se proporcionan las estimaciones de esfuerzo (en personas-mes) para cada tarea de la ingeniería en programación de cada función del software de CAD. Los totales horizontal y vertical proporcionan una indicación del esfuerzo requerido.

Los coeficientes de trabajo están asociados con cada tarea de la ingeniería en programación y forma parte de la fila "Coeficiente $" de la tabla. En este ejemplo se asume que los costos de trabajo para el análisis de requerimientos ($5.200/persona-mes) serán el 22% mayores que los costos para la implementación y la etapa de pruebas.

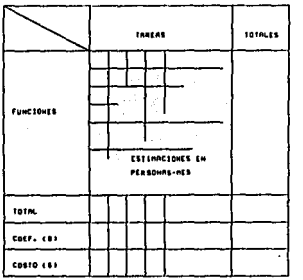


Fig. III.7. Desarrollo de una matriz de costos

TAREAS / FUNCIONES	ANALISIS DE REQUERIMIENTOS	DISEÑO	CODIFICACION	PRUEBA	TOTALES	
IUFC	1.0	2.0	0.5	3.5	7	
RG2D	3.0	10.0	4.5	3.5	21	
RG3D	2.5	15.0	8.0	11.0	31.5	
GED	3.0	6.0	3.0	4.0	15	
FVGC	1.5	11.0	4.0	10.5	27	
PCP	1.5	6.0	2.5	3	10	
MAD	4	14	5	7	30	
=TOTAL	14.5	61	29.5	50.5	192.5	← ESFUERZO ESTIMADO PARA TODAS LAS TAREAS
COEFICIENTE B	3200	4000	4250	4500		
COSTO $	15,400	252,000	110,625	227,250	760,275	← COSTO ESTIMADO PARA TODAS LAS TAREAS

* TODAS LAS ESTIMACIONES ESTAN EN PERSONAS-MES EXCEPTO CUANDO SE INDIQUE LO CONTRARIO.

Fig. III.8. Desarrollo de una matriz de costos

El costo y el esfuerzo total estimado para el software de CAD es de $708.000 y de 153 personas-mes, respectivamente. Comparando estos valores con los datos obtenidos utilizando la técnica de líneas de código se encuentra una variación en el costo del 7% y una variación del esfuerzo del 5%. Como se puede observar se ha conseguido un acuerdo extremadamente próximo.

Cuando las estimaciones son en extremo demasiado divergentes pueden a menudo ser debidas a una de dos causas:

- El alcance del proyecto no es entendido adecuadamente o ha sido malinterpretado por el planificador.
- Los datos de productividad utilizados en la estimación LDC o PF son inapropiados para la aplicación, obsoletos, o han sido mal aplicados.

III.3.3. Herramientas automáticas de estimación

Las herramientas automáticas de estimación permiten al planificador estimar costos y esfuerzos así como llevar a cabo análisis con importantes variables del proyecto tales como la fecha de entrega o la selección de personal. Aunque existen muchas herramientas automáticas de estimación, todas muestran las mismas características generales y todas requieren una o más de las siguientes clases de datos:

- Una estimación cuantitativa del tamaño del proyecto o de la funcionalidad.
- Características de la calidad del proyecto tales como la complejidad, la fiabilidad requerida, o el nivel crítico del negocio.
- Alguna descripción del personal de desarrollo y/o del entorno de desarrollo.

De estos datos, el modelo desarrollado por la herramienta automática de estimación proporciona estímaciones del esfuerzo requerido para completar el proyecto, los costos, la carga de personal y, en algunos casos, la agenda de desarrollo y el riesgo asociado.

A continuación se mencionan dos herramientas automáticas representativas:

SLIM, es un sistema automático de cálculo de costos, el cual esta basado en el modelo de estimación de Putnam[1], la programación lineal, la simulación estadística y las técnicas PERT (un modelo de planificación) para obtener estimaciones del proyecto de software. El sistema permite al planificador del proyecto realizar las siguientes funciones:

- Calibrar el entorno local de desarrollo de software interpretando los datos históricos proporcionados por el planificador.
- Crear un modelo de información del software desarrollado mediante las características básicas del software.
- Llevar a cabo un cálculo del tamaño del software.

Una vez que ha sido establecido el tamaño del software, SLIM calcula la desviación del tamaño, una indicación de incertidumbre de la estimación, un perfil de sensibilidad que indica la desviación potencial del costo y del esfuerzo y una comprobación de consistencia mediante datos obtenidos de sistemas de software de tamaño similar.

1 - Modelo descrito en la sección 3.8.2., Ingeniería del Software Roger S. Pressman.

ESTIMACS, es un "modelo de macro-estimación" que utiliza el método de estimación del punto de función, para acomodarse a variados proyectos y distintos factores de personal.

La herramienta ESTIMACS contiene un conjunto de modelos que permiten al planificador estimar:

- El esfuerzo del desarrollo del sistema.
- El costo y el personal.
- La configuración del hardware.
- El riesgo y
- Los efectos de la "cartera de desarrollo" (efectos de otro trabajo concurrente).

Utilizando los datos del modelo de esfuerzo de desarrollo del sistema. ESTIMACS puede desarrollar planes de personal y de costos utilizando "una base de datos a medida del ciclo de vida para proporcionar información de distribución del trabajo y de despliegues".

III.4. HERRAMIENTAS DE CONTROL DE AVANCE

La planificación temporal para proyectos de desarrollo de software puede ser vista desde dos perspectivas diferentes. En la primera, la fecha final de lanzamiento del sistema basado en computadora ya ha sido establecida, y la organización del software se ve forzada a distribuir el esfuerzo dentro del marco prescrito. El segundo enfoque de la planificación temporal del software asume que se han discutido unos límites cronológicos aproximados, pero que la fecha final es fijada por la organización del software.

La precisión en la planificación puede a veces ser más importante que la precisión en los cálculos de costos. En un entorno orientado al producto, los costos añadidos pueden ser absorbidos por la inflación o la amortización sobre un amplio número de ventas. Sin embargo, una falla de programación en la planificación, puede reducir el impacto de mercado, generar clientes insatisfechos y aumentar los costos internos por la existencia de problemas adicionales durante la integración del sistema.

Al enfocar la planificación temporal del proyecto de software se deben plantear ciertas cuestiones como son:

¿Cómo correlacionamos el tiempo cronológico con el esfuerzo humano?, ¿Qué tareas y que paralelismo se pueden esperar?, ¿Qué puntos de referencia se pueden utilizar para mostrar el progreso?, ¿Hay disponibles métodos de análisis de la planificación temporal y cómo se representa físicamente una agenda?.

La identificación de los requerimientos de tiempo del proyecto de software no permite conocer el número de días, semanas o meses que es necesario integrar al programa de tiempos del mismo. En la mayor parte de los proyectos de sistemas se utiliza tiempo adicional en las actividades de proyección. Las reuniones de la gerencia, revisiones del proyecto, educación y capacitación, interacción con los usuarios, tiempo de incapacidad por enfermedad, vacaciones, días de fiesta y la dificultad en obtener sistemas de cómputo, extienden el programa más allá de los estimados previos, de hecho, en los proyectos de sistemas grandes, el tiempo adicional necesario para estas actividades extiende el tiempo total de desarrollo de 50 a 100%.

Basandose en estos datos, un proyecto que se estime que requiera 250 días de programación, llevará en la práctica de 375 a 500 días de programación.

Un día de programación (algunas veces denominado persona-día) se utiliza para medir el número de días que un proyecto requerirá basandose en el trabajo que una persona

puede realizar durante un día. El número de personas que trabajan en el proyecto afecta el tiempo del calendario, aunque no de manera proporcional. Así que añadir más personal a un proyecto reduce el tiempo total de calendario, pero debe asignarse también el tiempo que no pertenezca al proyecto.

Por lo general se utilizan tres métodos para planear los requerimientos de tiempo de calendario; gráficas de barras, gráficas de punto de referencia y gráficas PERT (Proyect Evaluation and Review Technique: Técnicas de Evaluación y Revisión de Proyectos).

III.4.1. Gráficas de barras (Gráficas de Gantt)

La planeación más sencilla emplea gráficas de barras que muestran cada actividad en un proyecto de sistemas y el tiempo que requerirán, este método, fué desarrollado por Henry L. Gantt (las gráficas algunas veces se denominan gráficas de Gantt), utiliza barras para indicar el tiempo que se emplea en cada tarea como se muestra en el ejemplo de la Fig. III.9. El analista identifica primero cada tarea y estima el tiempo que es necesario para desempeñarla, cuando esta información se transfiere a la gráfica de barras, las tareas se listan de arriba a abajo en la parte izquierda de la gráfica, en el orden en que se llevarán a cabo. El tiempo de calendario se ilustra de izquierda a derecha. Una barra horizontal se marca en la gráfica para cada tarea, indicando cuando da principio y cuando se espera que termine. De esta manera, la primera tarea en la Fig. III.9. se ha planeado para que se realice en dos semanas.

Tres actividades separadas que se inician en la tercera semana requieren de dos a cuatro semanas cada una. La ausencia de una barra significa que no se asocia ninguna tarea con la actividad durante un periodo específico.

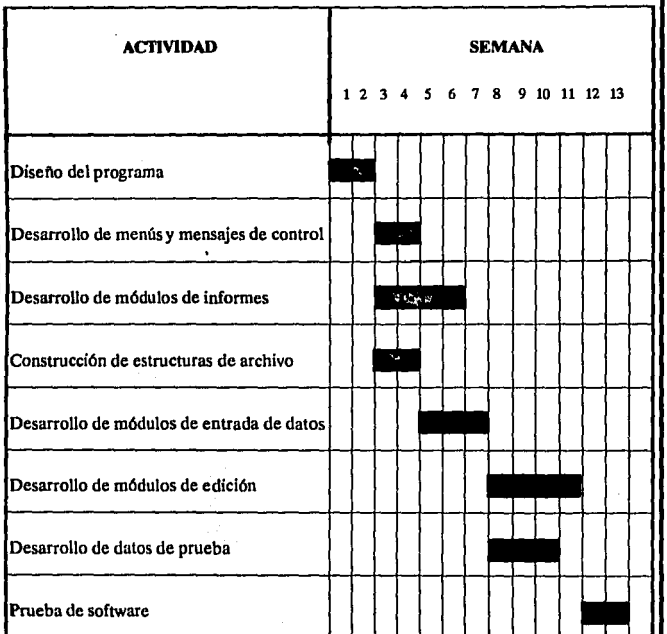


Fig. III.9. Plan de desarrollo de un sistema mediante una gráfica de barras.

Las gráficas de barras son manejaoles cuando el proyecto consiste en un número limitado de actividades o tareas, de otra manera, el tamaño de la gráfica se vuelve desproporcionado e incluye tantas barras que es difícil utilizar la información.

III.4.2. Gráficas de puntos de referencia (mili stoner).

Todos los proyectos tienen acontecimientos significativos denominados puntos de referencia, los cuales marcan puntos importantes en su desarrollo. Representan etapas difíciles que deben pasarse o tareas críticas que es necesario completar a tiempo, las gráficas de puntos de referencia ilustran los acontecimientos significativos al completar un proyecto y la secuencia en que se deben terminar.

Las gráficas de puntos de referencia ilustran los acontecimientos significativos al completar un proyecto y la secuencia en que se deben terminar. Difieren de las gráficas de barras porque representan puntos en que algo se completa y no tareas individuales que se pueden llevar a cabo; por ejemplo, al completar la puesta en marcha, los puntos de referencia de un sistema incluyen la llegada del equipo, instalación, terminación de la capacitación de los usuarios, conversión de archivos y el cambio al nuevo sistema.

El progreso de un proyecto se mide al comparar la situación de una actividad con la fecha del punto de referencia o de terminación; esto permite al gerente del proyecto mantenerse a tiempo. La desventaja de este método radica en la importancia total que se le da al tiempo y no a la interdependencia entre las tareas y acontecimientos de control de los costos del proyecto.

III.4.3. Gráficas PERT

El método más complejo para la planeación es el método PERT (Project Evaluation and Review Technique). Este método, desarrollado por la armada de Estados Unidos y Booze, Allen y Hamilton, una compañía consultora de gerencia, se ha utilizado en muchos proyectos complejos que requieren de una planeación y administración cuidadosa.

Aunque el mejor enfoque para el manejo de proyectos consiste en dividir el proyecto en partes pequeñas y manejables, existe el peligro de perder la visión global de todo el proyecto mientras se supervisan las tareas menores. Las actividades del proyecto son por lo general independientes; sin embargo, la interdependencia de las tareas no aparece en los métodos de gráfica de barras o de puntos de referencia. Las tareas críticas, las que se deben completar a tiempo y en una secuencia específica, tampoco son evidentes.

Un gerente de proyectos, desea indicar las actividades individuales y el tiempo necesario para cada una, mostrar la interrelación de las actividades, identificar la secuencia adecuada, dar estimaciones de tiempo, separar áreas específicas en donde los problemas esenciales o los retrasos pueden ocurrir, y tener un método para la verificación del progreso en el proyecto. Puede ser necesario saber, por ejemplo, cuál es el efecto del retraso de cierta actividad en todo el proyecto o para que actividad existe la menor tolerancia para un retraso. Si se desarrollan adecuadamente, las gráficas PERT proporcionan esta información.

Los proyectos consisten en acontecimientos y actividades, en la gráfica se utilizan círculos (nodos) y caminos (rectas o arcos) para representar la interrelación de las actividades del proyecto (fig. III.10.). Los nodos representan acontecimientos y los caminos muestran las actividades que se requieren para adelantarse de un acontecimiento al otro. Los números sobre las líneas indican el tiempo necesario para llevar a cabo cada actividad.

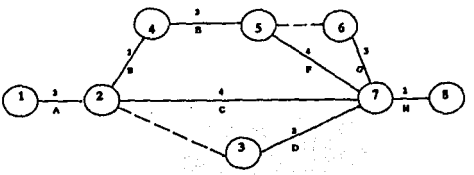


Fig. III.10. Gráfica PERT

La gráfica PERT tiene un gran valor cuando un proyecto se planea y se diseña. Cuando se da término a la gráfica de relaciones se estudia para determinar la ruta crítica, es decir, el camino que es necesario llevar desde el principio hasta el fin y por el cual el tiempo total requerido será mayor que para cualquier otro camino (como se muestra en la linea gruesa de la fig. III.11.). Si las actividades a lo largo de esta ruta no se termina a tiempo, todo el proyecto se retrasará.

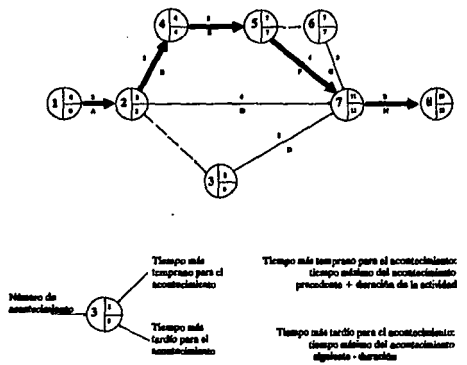


Fig. III.11. Tiempos de actividad en la red PERT

La gráfica PERT muestra también las interdependencias de las tareas y ayuda a responder tres preguntas comunes de la gerencia de proyectos:

- ¿Qué otras actividades deben preceder o se deben completar antes del inicio de una actividad específica?

- ¿Qué otras actividades se pueden llevar a cabo en tanto una actividad específica se encuentra en progreso?
- ¿Qué actividades no se pueden iniciar hasta después de terminar una actividad específica?

Para desarrollar una red PERT para un proyecto de sistema de información, se deben identificar primero las tareas y los tiempos relacionados con cada actividad, el tiempo de cada actividad es la duración. A continuación, se debe identificar la secuencia de actividades y los lugares en donde las tareas específicas deben preceder a otras y en donde ciertas actividades pueden ocurrir de manera simultánea con otras anotadas. Esta información se muestra en la red que se ha desarrollado en la fig. III.11. las rectas representan actividades de la A a la H y el número en cada línea constituye el número de semanas que el gerente del proyecto espera que llevará esa actividad (es, decir la duración de la actividad).

En este ejemplo, los acontecimientos (eventos) 2-4, 2-7 y 2-3 no se pueden iniciar antes de que la tarea A se haya terminado, como se indica por la ubicación de los acontecimientos en la red. La línea punteada entre los acontecimientos 2 y 3 significa que no se ha asociado ningún tiempo a la línea de puntos y no posee una identificación de tareas (una letra o un nombre). Cuando existe una dependencia entre dos acontecimientos pero no se consumen tiempo o recursos, se denomina actividad nula. las actividades nulas pueden, por lo tanto, utilizarse para conectar actividades paralelas (las actividades C y D pueden ocurrir de manera paralela). Antes de que la tarea D de la Fig. III.11. pueda iniciarse, el acontecimiento 2 debe terminarse.

El siguiente paso consiste en analizar el programa de tiempos del proyecto, se desea saber:

- ¿qué tan pronto puede iniciarse el acontecimiento? y
- ¿cuánto puede tardarse en iniciar éste sin afectar el programa general del proyecto.?

El tiempo más temprano para el acontecimiento (denominado TMT) es cero para el primer acontecimiento, para los otros es *La suma más alta de la duración de la actividad y el TMT de cualquier acontecimiento inmediatamente precedente.* Por ejemplo, cuatro actividades preceden de inmediato el acontecimiento 7. El TMT se computa mediante el análisis del TMT y el tiempo de duración para cada acontecimiento que le precede.

Ya que el acontecimiento 7 no puede iniciarse hasta que todos los acontecimientos precedentes se hayan completado, lo que interesa es el factor de tiempo mayor, que en este caso es el 11. La Figura III.11. muestra el TMT para toda la red. Una notación utilizada con frecuencia muestra el TMT en la parte superior derecha del nodo y el TMt (tiempo más tardío) en la parte inferior derecha.

El tiempo más tardío del acontecimiento es el tiempo límite en el cual el mismo puede dar inicio sin retrasar el proyecto. Para determinar este tiempo es necesario trabajar retrospectivamente en la red iniciando desde la derecha. El tiempo más tardío constituye la *diferencia menor entre el TMt del acontecimiento terminal menos el tiempo de duración de la actividad.* Por ejemplo, para determinar el TMt para el acontecimiento 7, el TMt del nodo siguiente, el número 8 en la figura III.11., se ha designado como 13. Si se resta el

tiempo de duración, el tiempo del acontecimiento o evento máximo se determina como 13 - 2 = 11. El número en la parte derecha inferior en el nodo 7 se muestra como 11.

La *ruta crítica* es el conjunto de actividades que los gerentes deben supervisar más cercanamente. Identificar los acontecimientos que se deben iniciar y terminar a tiempo y que requieren no más que el tiempo de duración estimado; de otra manera todo el proyecto sufrirá un retraso.

En la figura III.11. se ilustra la ruta crítica en líneas más gruesas. Se determinó al vincular todos los nodos en donde los TMT y TMt son iguales, lo que significa que no hay posibilidades de cambio o de desviación, es decir, no hay tiempo de tolerancia u holgura.

El tiempo de holgura que se relaciona con un acontecimiento se puede ilustrar formalmente restando el tiempo de duración y el TMT del nodo inicial del TMt del nodo terminal. Por ejemplo, para el acontecimiento 3-7, el tiempo de holgura es de 7 semanas (11-2-2=7), lo que significa que la actividad 3-7 puede iniciarse en cualquier momento de la segunda hasta la novena semana y, si el tiempo de duración de 2 semanas se mantiene, el proyecto estará a tiempo.

Las gráficas PERT son muy útiles en el diseño, como se ha indicado, pero son muy difíciles de mantener y actualizar. Por esta razón es conveniente utilizar una combinación de gráficas: gráficas PERT para planear el desarrollo e ilustrar las interdependencias, y gráficas de barras para mostrar los programas de calendario.

CAPITULO IV

ANALISIS Y ESPECIFICACION ESTRUCTURADA

INTRODUCCION.

El análisis estructurado se enfoca en especificar *Que* es lo necesario que haga el sistema o que se debe obtener con la aplicación . No establece de que manera se deben cumplir las demandas o cómo debe ser puesta en marcha la aplicación. Más bien permite que los individuos vean lo que el sistema debe hacer independientemente de las restricciones físicas. Después se puede elaborar un diseño físico que va a ser efectivo en cuanto a costo para la situación en la cual va a ser usado.

El análisis estructurado utiliza modelos gráficos debido a que éstos muestran a los gerentes y al personal que no es de sistemas detalles del sistema sin tener que presentar el manual o los procesos de cómputo, archivos de cinta o discos o programas y procedimientos operativos. Si se seleccionan símbolos y notación apropiados, virtualmente cualquiera puede seguir la forma en que las componentes en el sistemas se ajustan una a otra.

IV.1. EL DIAGRAMA DE FLUJO DE DATOS

Un diagrama de flujo de datos es una representación en forma de red de un sistema. El sistema puede ser automatizado, manual o mixto. El diagrama de flujo de datos presenta al sistema en términos de sus piezas componentes esenciales e indicando las interfaces entre sus componentes. Los diagramas de flujo de datos también son conocidos como: gráficas de flujos de datos y diagramas de burbuja.

Los diagramas de flujo de datos muestran las características lógicas de las aplicaciones; es decir, señalan qué ocurre y cuándo, pero no establece cómo; en otras palabras, los detalles físicos son métodos de almacenaje, dispositivos de entrada o componentes de la computadora (como se definen en los diagramas de flujo tradicionales), o sea, no son aspectos principales para los analistas durante la parte inicial del análisis; sólo después de entender los componentes lógicos, deben estudiarse los componentes físicos.

IV.1.1. Características de los Diagramas de Flujo de Datos.

El análisis del flujo de los datos permite a los analistas aislar las áreas de interés, estudiarlas y examinar los datos que entran al proceso y ver como se modifican cuando salen del mismo. Conforme los analistas recopilan hechos y detalles, su creciente conocimiento del proceso los lleva a realizar preguntas sobre partes específicas del proceso, que a su vez los conduce a una investigación adicional.

Una investigación de sistemas elaborada en forma amplia produce conjuntos de muchos diagramas de flujos de datos, algunos de los cuales proporcionan una visión general de los procesos principales, y otros dan un mayor detalle para señalar elementos de datos, almacenamiento de estos y etapas del proceso para componentes específicos de un sistema mayor. Si los analistas desean revisar posteriormente el sistema en su totalidad, utilizan

los diagramas generales. Sin embargo, si estan interesados en estudiar un proceso en particular, utilizan los diagramas de flujo de datos de los procesos de menor nivel.

Los niveles de diagramas de flujo de datos se pueden comparar con mapas de carreteras y calles, que se utilizan cuando se viaja en una área no familiar. En un viaje largo, primero se usa un mapa a nivel nacional que indica las principales carreteras y ciudades. Conforme se acerca a la ciudad que se visita, se necesita un mapa más detallado que muestre los principales lugares de la ciudad y como llegar a éstos. Después de que se llega al área de la ciudad que se desea, es muy útil un mapa detallado de calles que señale los sitios más importantes, como puentes y edificios. Por lo tanto, esta información es esencial para encontrar las direcciones correctas de las calles; no obstante, no es útil cuando se inicia el viaje y se empieza a tener problemas para obtener una orientación.

Los diagramas de flujo de datos se usan de la misma forma. Se desarrollan y utilizan de manera progresiva, que va de lo general a lo particular, para el sistema de interés (fig. IV.1). Los diagramas de flujo de datos no serán útiles si no se dibujan en forma apropiada o son difíciles de manejar. Cada nivel inferior se define utilizando de tres a siete procesos para extender un proceso de mayor nivel. Utilizar más de siete provoca que el diagrama sea difícil de manejar. Los diagramas de flujo de datos son más fáciles de leer si la descripción de un proceso de un nivel puede dibujarse en una sola hoja de papel. La guía para utilizar de tres a siete procesos para extender los procesos de mayor nivel deberá permitir que la descripción ocupe una sola hoja de papel. Cuando hay demasiados círculos se deben juntar varios en uno solo y descomponerlo en otro nivel.

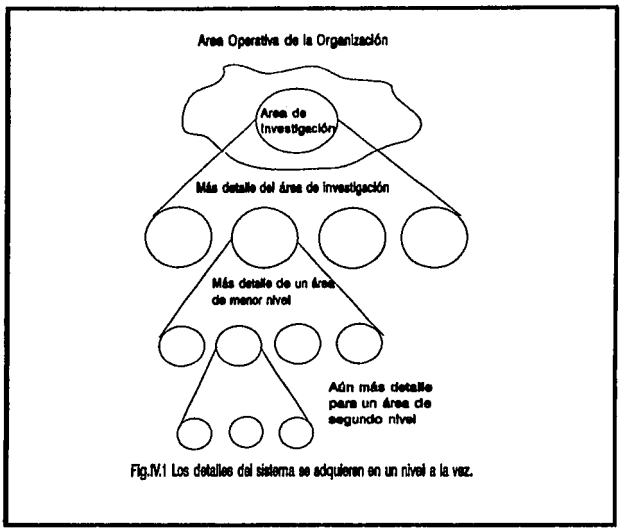


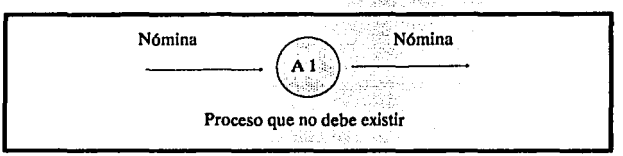
Fig.IV.1 Los detalles del sistema se adquieren en un nivel a la vez.

En el diagrama, cada componente de flujo de datos se etiqueta con un nombre descriptivo. Los nombres del proceso se identifican aún más con un número que se utilizará para propósitos de identificación. El número asignado para un proceso específico no representa la secuencia del mismo; es estrictamente para identificación y tendrá otro valor más cuando se estudien los componentes que conforman un proceso específico.

IV.1.2. Notación.

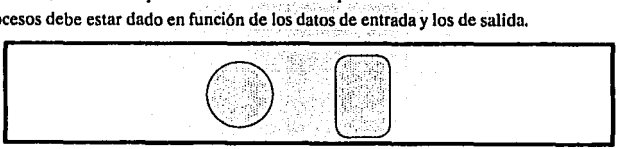
Los diagramas de flujo de datos lógicos pueden completarse al utilizar solamente cuatro notaciones sencillas:

1.- Flujo de Datos.- Es un datoducto por el que fluyen paquetes de composición definida. Los flujos de datos representan conjuntos conocidos de datos a los que se les puede poner un nombre; si no se les puede poner, es un error de representación. En un mismo diagrama de flujo de datos no puede existir dos flujos con el mismo nombre, si existiera esto quiere decir que no hubo transformación en el proceso y por lo tanto no debe existir el proceso.

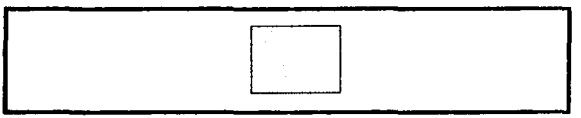


Si el nombre en el flujo de datos no representa un dato debe ser un comando o un evento.

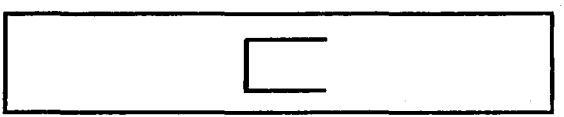
2.- Procesos.- Es una transformación de los flujos de datos que entran, en los flujos de datos que salen; es la única parte donde los datos se pueden transformar. El nombre de los procesos debe estar dado en función de los datos de entrada y los de salida.



3.- Origen o destino de los datos o terminador.- El origen externo o destino de los datos, que pueden ser individuos, programas, empresas u otras entidades, interactúan con el sistema pero están fuera de su límite.



4.- Depósitos o datos almacenados.- Es un receptor de información en el tiempo de vida del sistema. Solo puede contener flujos netos (los datos no se pueden transformar dentro de un depósito). El número de datos de entrada debe ser igual a los de salida.



En la fig. IV.2 se muestra un diagrama de flujo de datos típico, donde podemos visualizar la simbología.

En la mayoría de los casos, los nombres del flujo de datos deben aclarar la relación entre los múltiples flujos de datos que entran y salen de los procesos. Algunos analistas utilizan notaciones adicionales para aclarar los casos en donde existen múltiples flujos de datos, como se muestra en la fig. IV.3.

El ⊕ (que representa " Y ") indica que ocurren ambos flujos de datos, mientras " * " (que representa " O ") muestra que uno de varios flujos de datos se esta llevando acabo.

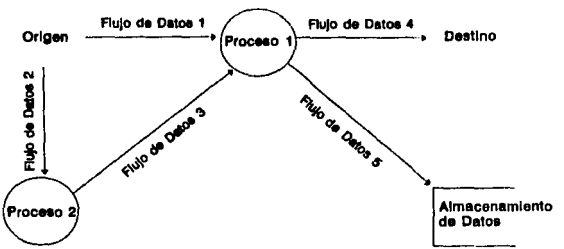


Fig. IV.2 Diagrama de Flujo de Datos

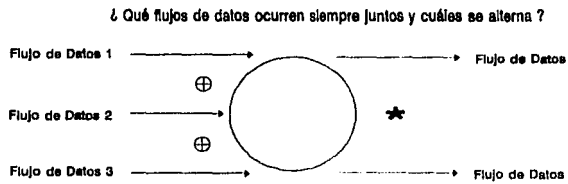


Fig. IV.3 Notaciones Adicionales

IV.1.3. Desarrollo de los diagramas de flujo de datos.

Para ser útiles e informativos, los diagramas de flujo de datos deben dibujarse en forma apropiada. A continuación se muestra cómo hacerlo.

IV.1.3.1. Guía para Dibujar Diagramas de Flujo de Datos.

Teniendo ya las características y convenciones sobre el DFD, el problema es ahora cómo dibujar uno de ellos. Los siguientes puntos forman una buena aproximación de propósito general para dibujar un DFD:

- Identifique todos los flujos de datos que representen entradas y salidas netas del sistema. Dibujelos en el exterior de su diagrama.
- Trate de relacionar las entradas con las salidas comenzando de las salidas hacia las entradas.
- Etiquete todos los flujos de datos en las interfases cuidadosamente.
- Etiquete las burbujas en términos de sus entradas y salidas.
- Ignore inicialización y terminación.
- Omita detalles de trayectorias para errores triviales (por ahora).
- No trate de mostrar flujo de control o información de control.
- Ya esta usted preparado para comenzar.

IV.1.3.2. Identificando las Entradas y Salidas Netas.

La primera parte del análisis consiste en determinar quiénes son los usuarios del sistema, quiénes proporcionarán información y quiénes son los que requieren información del sistema.

Una vez que se han identificado estas entidades, es conveniente visualizar aquellas que solamente proporcionan información es decir, son fuentes de información o bien, a los que solamente consumen información (son sumideros). También, es conveniente establecer si una entidad procesa información y si ésta queda dentro o fuera del contexto del sistema.

Esto nos permitirá establecer la frontera del sistema, así como sus interfases con el mundo exterior. (fig. IV.4)

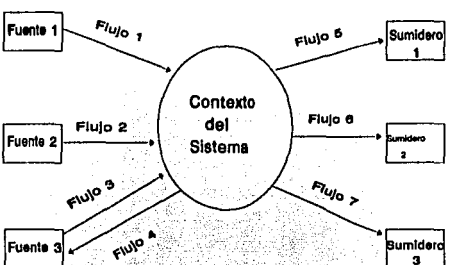


Fig.IV.4 Primer acercamiento al sistema

IV.1.3.3. Determinación del Cuerpo del Diagrama de Flujo de Datos.

En casi todos los proyectos, el primer uso del Diagrama de flujo de Datos es para documentar el área de trabajo de los usuarios. Al hacer esto, debe describirse lo que ellos son. Consecuentemente, un mismo usuario puede aparecer dos veces o más en el diagrama, pero efectuando diferentes procesos. Sus flujos de datos serán paquetes de datos a los cuales el usuario les ha asignado un nombre. Sus procesos serán segmentos del trabajo que se efectúa en esa área.

Concentremonos primero en los flujos de datos. Tenemos los conductos más grandes. Supongamos que se ha identificado un conjunto significativo de información que el usuario trata como una unidad (es decir, viajan y son procesados todos los objetos de él como un todo).

Una vez identificados los flujos de datos, trate de conectarlos con los flujos de datos, de la periferia (frontera); colocando burbujas en la parte que sea necesario efectuar un trabajo para transformar un conjunto de datos y otros flujos que fluyen a otros procesos. No trate de darle nombre a estos procesos; dejelos por el momento en blanco.

Una vez que se tienen los procesos en blanco, trate de imaginar como son utilizados los flujos dentro del proceso. Si existe alguna duda, verifique con el usuario y si el proceso es demasiado complejo, trate de repararlo en dos, tres, cuatro o más procesos, identificando los nuevos flujos de datos que fluyen entre ellos.

Para cada flujo de datos, responda a la pregunta: Que se necesita hacer para construir ese objeto? Si es un MANIFIESTO DE PASAJEROS, por ejemplo, usted puede determinar que esta hecho de BOLETOS-COLECTADOS-DE-AVION más la LISTA-ORIGINAL-DE-VUELO más algunas otras formas. De donde provienen estos componentes?, Puede cualquiera de los flujos de datos de entrada ser transformados en cualquiera de estos?, Qué procesos son necesarios para efectuar las transformaciones?

Incluya archivos en su Diagrama de Flujo de Datos para representar los diferentes almacenes de datos que el usuario utilice. Asegurese de conocer el contenido de cada archivo con el suficiente detalle para poderse figurar el flujo hacia dentro y afuera del mismo.

Recuerde que puede regresar a modificar el contexto de la frontera. Puede haber olvidado una entrada que es requerida como una componente clave en sus flujos; añádalos. Asimismo, puede tener un flujo de datos que desaparece, es decir, no se utiliza dentro del contexto del sistema; quitelo.

Puede en algunos casos presentarse una situación similar a la de la figura IV.4, donde su contexto incluye dos redes, desconectadas, una de las cuales esta completamente separada del verdadero dominio que le interesa. En este caso se puede eliminar la red extra.

IV.1.3.4. Etiquetando los Flujos de Datos.

Los nombres que usted seleccione para sus flujos de datos tendrán un gran efecto en la capacidad de comprensión de su diagrama de flujo de datos. Para ello:

- Asegurese de dar un nombre a cada flujo de datos. Aquellos que deje sin nombre son invariablemente los que son imposibles de nombrar (debido a su pobre particionamiento). Hay que separarlos en dos o más flujos.

- Asegurese que el nombre sea honesto, es decir, aplíquelo al flujo completo, no a su mayor componente.

- Evite nombres demasiado genéricos como "datos" o "información ".

- Sea cuidadoso de no agrupar objetos diferentes en un mismo flujo de datos cuando ellos no puedan ser tratados como un todo.

IV.2. EL DICCIONARIO DE DATOS.

El Diccionario de Datos es un parte integral de la Especificación Estructurada; sin él, los Diagramas de Flujo de Datos son únicamente bonitas figuras que dan idea de lo que hace un sistema. Cuando cada elemento del Diagrama de Flujo de Datos haya sido rigurosamente definido, podremos decir que el conjunto constituye una "Especificación". El conjunto de todas las definiciones rigurosas de los elementos del Diagrama de Flujos de Datos es el Diccionario de Datos.

El papel más importante de un Diccionario de Datos, es el de proporcionar un lugar único en el que se tenga las definiciones de los términos que no se conocen.

Un Diccionario de Datos almacena información de referencia cruzada, la cual es muy importante, ya que permite que el diseñador determine con rapidez donde se usa un cierto conjunto de datos.

Considerado como un instrumento de diseño, un Diccionario de Datos posee las siguientes características útiles adicionales:

- Disminuye la redundancia y la incongruencia de los datos facilitando la comunicación entre el usuario y diseñadores respecto al almacenamiento y al empleo de los datos.

- Ofrece un depósito central de información de diseño.

- Permite que los diseñadores puedan determinar cual será el efecto de un cambio en los requerimientos de los datos de una aplicación sobre otras aplicaciones.

Los Diagramas de Flujo de Datos y el Diccionario de Datos son considerados un conjunto. Sin un Diccionario de Datos, los diagramas carecen de vigor; sin los diagramas, el Diccionario pierde sentido. La correlación es como sigue:

> Existe una entrada en el Diccionario de Datos para cada flujo que aparece en el conjunto del Diagrama de Flujo de Datos. Existe una entrada en el DD para cada archivo referenciado en cualquier diagrama del conjunto. Existe una entrada en DD para cada función primitiva en el conjunto.

IV.2.1. Consideracciones de Implementación.

Existen tres posibilidades de uso común hoy en día:

- Procedimientos totalmente manuales, haciendo uso de archivos de tarjetas indexadas, libros de notas, etc.
- Procedimientos totalmente automatizados, involucrando uno de los paquetes disponibles de DD (posiblemente modificado para hacerlo más adaptable a los requerimientos especiales de la fase de análisis).
- Combinación de procedimientos manual y automatizado, tomando ventajas de cualquier facilidad disponible en la organización (procesadores de palabras, generadores de reportes,etc.)

La descripción de los datos establece brevemente lo que representan en el sistema. Las descripciones de deben escribir suponiendo que la gente que los lea no conoce nada en relación al sistema. Deben evitarse términos especiales o argot, todas las palabras deben ser entendidas por el lector. El uso de los alias debe evitar confusión, un diccionario de datos significativo incluirá todos los alias.

La redundancia son los casos donde aparece inecesariamente la misma información en sitios diferentes. La redundancia tiene dos orígenes principales:

- El usuario maneja sinónimos.
- Mala planeación y diseño en el suministro de llaves.

Con el propósito de evitar redundancia entre los elementos de la especificación estructurada, es esencial establecer guías que indiquen que clases de información, atañe a cada porción de la especificación:

- La información sobre la composición de datos (cuales son sus componentes y como se interrelacionan) va en el diccionario de datos.
- La información sobre el contenido y proceso de datos (como son establecidos sus valores) va en la descripción del proceso.
- La información sobre la ruta de datos va en el diagrama de flujo de datos.

IV.2.2. Notación.

Los símbolos que se utilizan en un diccionario de datos son los siguientes:

= Significa "Es equivalente a"

+ Significa "Y"

[] Significa "O", es decir, seleccione una de las opciones encerrradas en los paréntesis.

{} Significa ITERACIONES de los componentes encerrados.

* Comentarios

( ) Significa que el componente encerrado es opcional.

| Significa "O".

Las iteraciones pueden ser anotadas con límites superiores e inferiores. Algunos analistas escriben los límites como superíndices (límite superior) y subíndices antes del paréntesis del lado izquierdo.

Otros ponen el límite inferior a la izquierda del paréntesis que se abre y el límite superior a la derecha del paréntesis que se cierra.

$$\overset{5}{\underset{1}{\{}}\ X\ \} \text{ es lo mismo que } 1\ \{\ X\ \}\ 5$$

Por ejemplo:

$$\text{Reservación} = \text{Nombre\_Huesped} + \text{Domicilio} + (\text{Empresa}) + \left\{ \begin{matrix} \text{Modo de Pago} \\ \text{Anticipo} \end{matrix} \right\}$$

El diccionario de Datos Reservación esta formado por el nombre del huesped y el domicilio y opcionalmente puede venir de una empresa y el modo de pago o el anticipo que se dió al momento de hacer la reservación.

Si ponemos:

Reservación = { Nombre_Huesped + Domicilio + ...}

quiere decir que Reservación esta formado por N repeticiones, es decir, estamos hablando de un archivo.

Cuando nos estamos refiriendo a un archivo, y éste tiene una llave de acceso, la forma de señalarla es subrayando ese campo:

Empleado = { # de Filiación + Datos Generales + Percepciones }

Otro ejemplo puede ser el de la variable:

Estado_Civil = [Soltero | Casado | Viudo | Divorciado | Unión Libre]

que significa que Estado_Civil esta formado por cualquiera de esas opciones.

Los Datos Primos son datos que ya no estan compuestos por nada (datos primitivos), son datos que hablan por si solos, es decir al especificarlos se dice que están compuestos por si mismo (fecha = fecha). Una variable puede requerir descomposiciones sucesivas hasta llegar a los datos primos (datos primitivos o elementales).

IV.3. ARBOLES DE DECISION.

Un árbol de decisión es un diagrama que presenta condiciones y acciones en forma secuencial y, por lo tanto, muestra qué condiciones considerar inicialmente y cuales después, etc., también es un método que muestra la relación de condición y sus acciones permisibles. El diagrama se ve como las ramas de un árbol, de ahí el nombre de árbol de decisión (fig. IV.5).

La raíz de un árbol, en la parte izquierda del diagrama, es el punto inicial de la secuencia de decisión. La rama específica que debe seguirse, depende de las condiciones que existan y de la decisión que se tenga que tomar. Así, se va avanzando de derecha a izquierda haciendo una serie de decisiones sobre una rama en particular. Si se sigue cada punto de decisión se llega al otro grupo de decisiones que deben considerarse; por lo tanto los nudos del árbol representan condiciones que indican que debe hacerse una determinación sobre cuál condición existe antes de que pueda seleccionarse la siguiente ruta. En el lado derecho del árbol, se enlistan las acciones que deben tomarse y que dependen de la secuencia de condiciones que se sigue.

Desarrollar árboles de decisión tienen ventajas para los analistas en dos aspectos: primero que nada, la necesidad de describir formalmente las condiciones y acciones obliga a los analistas a identificar las decisiones actuales que deben realizarse. Los árboles de decisión también obligan a los analistas a considerar la secuencia de las decisiones.

Si los árboles de decisión se realizan después de terminar el análisis de flujo de datos, los datos críticos ya pueden definirse en el diccionario de datos. Si los árboles de decisión se utilizan solos (rara vez se hace esto), los analistas deben tener la certeza de identificar en forma precisa cada elemento que se necesita para tomar una decisión. El formato del diccionario de datos, es útil para enlistar y describir los elementos dato conforme se identifican y entienden.

Los árboles de decisión no siempre pueden ser la mejor herramienta para el análisis de decisiones. Si el sistema es tan complejo que existen muchas secuencias de etapas y combinaciones de condiciones, el analista debe evitar los árboles de decisión porque su tamaño será demasiado grande. Un gran número de ramas y demasiadas rutas, a través de las cuales se puede avanzar, dificultará más que ayudar al análisis. El analista puede no ser capaz de determinar qué políticas o prácticas del negocio guían la formulación de las decisiones específicas. Cuando surgen estos problemas, se deben considerar las tablas de decisión.

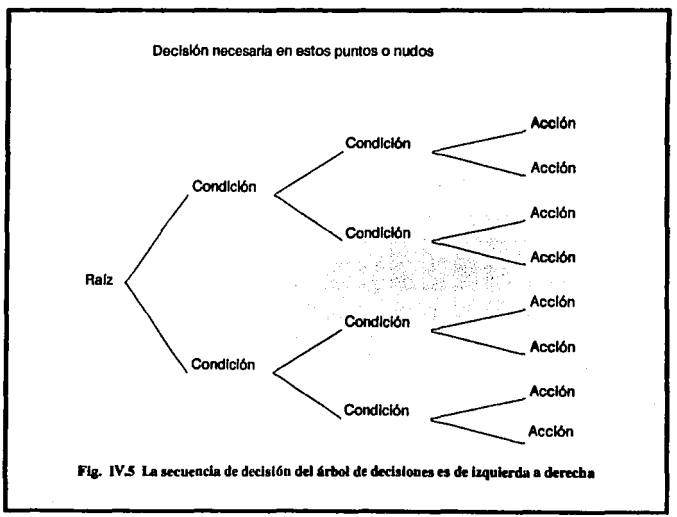


Fig. IV.5 La secuencia de decisión del árbol de decisiones es de izquierda a derecha

IV.4. TABLA DE DECISION.

Una tabla de decisión es una matriz de renglones y columnas más que un árbol y muestra condiciones y acciones. Las reglas de decisión también se incluyen en la tabla y establecen qué procedimiento seguir cuando existan ciertas condiciones (fig. IV.6).

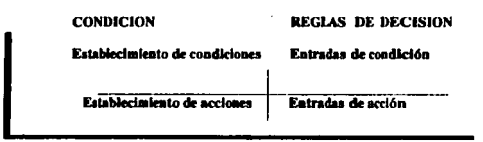


Fig. IV.6 Forma general de una tabla de decisión

La tabla de decisión se conforma de cuatro secciones: establecimiento de las condiciones, entradas de las condiciones, establecimiento de las acciones y entradas de acción. La sección de establecimiento de condiciones identifica las condiciones importantes. Las entradas de condición señalan qué valor, si es que hay alguno, aplica para una condición en particular. El establecimiento de acciones enlista el conjunto de todas las etapas que pueden llevarse a cabo cuando se cumple una condición. Las entradas de acción muestran qué acciones específicas, dentro del conjunto, hay que realizar cuando son verdaderas ciertas condiciones o combinaciones de las mismas. Algunas veces se añaden notas en la parte inferior de la tabla para indicar cuándo utilizarla o para distinguirla de otras tablas de decisión.

Las columnas del lado derecho de la tabla, que son las condiciones y acciones que se unen, forman las reglas de decisión. Una regla de decisión establece las condiciones que debe satisfacerse para que se lleve a cabo un conjunto de acciones en particular. Nótese que se desechó la secuencia del orden en la cual se examinaban las condiciones; esto fue necesario con los árboles de decisión. La regla de decisión incorpora todas las condiciones que deben ser verdaderas y no sólo una condición a la vez.

IV.4.1. Construcción de tablas de decisión.

Para desarrollar tablas de decisión, los analistas deben utilizar los siguientes pasos:

1.- Determinar los factores más relevantes y así considerarlo al tomar una decisión. Cada condición seleccionada debe tener la capacidad de ocurrir o no ocurrir; no son posibles medias ocurrencias.

2.- Determinar los pasos o actividades más factibles bajo condiciones variables (no solamente las condiciones actuales). Esto identifica las acciones.

3.- Estudiar las combinaciones de condiciones que son posibles. Para cada número N de condiciones existen un 2^N de combinaciones posibles que pueden considerarse. Por ejemplo, para tres condiciones existen ocho combinaciones; $2^3 = 8$. Para 4, $2^4 = 16$ combinaciones son posibles y pueden incluirse en la tabla.

4.- Completar la tabla con las reglas decisión. Existen dos formas para completar la tabla:

El primer método y el más largo es completar los renglones de condición, con un valor de sí o no para cada combinación posible de condiciones, es decir, completar la primera mitad del renglón con una S y la segunda con una N. El siguiente renglón debe llenarse de forma alternada con S y una N en el 25% del renglón: por ejemplo, 25% S, 25% N, 25% S y 25 S. Repítase este proceso. Llénese cada renglón que quede en forma alternada con S y N y divídase por potencias crecientes de dos. (Es decir $2^2=4$

para el segundo renglón, 2^3 para el tercer renglón hasta que se llegue a 2^N para el último renglón N donde N es el número de condiciones.)

El otro método que completa la tabla tiene que ver con una condición a la vez, y se añade a la tabla para condición adicional, pero no añade combinaciones duplicadas de condiciones y acciones:

- Establézcase la primera condición y acciones permisibles.
- Añadase la segunda condición duplicando la primera mitad de la matriz llenando con valores diferentes de S y N de la nueva condición en ambas mitades de la matriz expandida.
- Repítase el paso anterior para cada condición adicional.

5.- Llénese las entradas de acción con una X para señalar las acciones que deben llevarse a cabo, y dejarse las celdas en blanco o enmárquese con un "." para demostrar que no aplica ninguna acción en este renglón .

6.- Examínese la tabla en lo que se refiere a reglas redundantes o contradicciones entre éstas.

Estas simples guías no sólo ahorran tiempo al construir una tabla de decisión, a partir de la información recabada durante una investigación, sino también ayudarán a señalar dónde falta información, dónde las condiciones no importan dentro de un proceso o en dónde existen relaciones importantes o resultados que otros no habían detectado o no considerado. En otras palabras, utilizar las tablas de decisión puede producir un análisis más completo y exacto.

IV.4.2. Verificación de las tablas de decisión.

Después de construir cada tabla, los analistas verifican que esté correcta y completa para asegurarse que la tabla incluya todas la condiciones, junto con las reglas de decisión que las relacionan con las acciones. Los analistas también deben examinar la tabla en cuanto a redundancia y contradicciones.

Las tablas de decisión pueden convertirse en algo muy grande y poco manejable, si se les permite crecer en una forma incontrolada. Quitar las entradas redundantes ayudará a manejar el tamaño de la tabla. La redundancia se presenta cuando ambos puntos de los siguientes son verdad: 1) dos reglas de decisión son idénticas excepto por una condición y 2) las acciones para las dos reglas son idénticas.

Las reglas de decisiones se contradicen cuando dos o más reglas tienen el mismo conjunto de condiciones y las acciones son diferentes. (Recuérdese que si son iguales, entonces esto es una redundancia.)

Las contradicciones significan ya sea que la información del analista está incorrecta o que hay un error en la construcción de la tabla. Muchas veces, sin embargo, la contradicción es el resultado de que el analista haya recibido información diferente de diversos individuos sobre cómo toman sus decisiones. Una decisión específica puede tomarse al utilizar reglas diferentes. Encontrar este tipo de cosas puede ser muy útil para cualquier analista que esté tratando de mejorar una situación de decisión.

IV.4.3. Tipo de entradas de tablas.

IV.4.3.1. Forma de entrada limitada.

La estructura básica de la tabla, utilizada en los ejemplos anteriores, consistente solamente de S,N y "-", se llama forma de entrada limitada. Esta es una de las que más se utilizan. También existen otras dos formas ampliamente usadas.

IV.4.3.2. Forma de entrada Extendida.

La forma de entrada extendida reemplaza a las S y N con entradas de acción que la señalan al lector cómo tomar la decisión. En este formato, los establecimientos de condición y de acción no están completos; esta es la razón por la cual las entradas contienen más detalles que una S y N.

IV.4.3.3. Forma de entrada mixta.

Los analistas pueden preferir combinar las características de ambas formas de tabla, tanto la de entrada limitada como extendida en una misma tabla. Generalmente, sólo se debería utilizar una forma en cada sección de la tabla, pero entre las secciones de condición y de acción puede utilizarse cualquier forma.

IV.4.3.4. Forma de en caso contrario.

Todavía puede permitirse otra variación en las tablas de decisión para omitir reglas repetitivas a través de reglas EN CASO CONTRARIO. Para construir una tabla de

decisión con formato de EN CASO CONTRARIO, especifíquese las reglas con entradas de condición para cubrir todas los conjuntos de acción, excepto para una cuya regla se seguirá cuando ninguna de las otras condiciones en forma explícita sea verdadera. Esta regla está en la columna final del lado derecho, es decir, en la columna de EN CASO CONTRARIO. Si ninguna de las otras condiciones se da, entonces se sigue la regla de decisión de EN CASO CONTRARIO. La regla de EN CASO CONTRARIO elimina la necesidad de repetir condiciones que guían a las mismas acciones.

IV.4.4. Tablas múltiples.

Las grandes tablas de decisión pueden convertirse en algo difícil de manejar; por lo tanto, los analistas deben controlar el tamaño de la tabla. La forma de EN CASO CONTRARIO es una forma de manejar el tamaño. Una segunda manera es utilizar las tablas de decisión múltiples ligadas entre sí. Dependiendo de las acciones seleccionadas en la primera tabla, las acciones adicionales se explican por una o más tablas adicionales; cada tabla adicional proporciona mayores detalles sobre las acciones que hay que tomar. Las tablas múltiples también permiten a los analistas establecer las acciones repetitivas que deben presentarse, después de que se ha tomado alguna decisión y se continuará hasta que se alcance cierta condición.

Para utilizar este método los analistas construyen tablas de decisión por separado, que cumplen con los requerimientos normales y tratan con una decisión específica; estas se unen entre sí en una forma jerárquica: una tabla descendente de lo general a lo particular contiene las principales condiciones, las cuales, cuando se seleccionan, determinan que acciones adicionales y a que tablas hay que referirse para mayor detalle. Una instrucción de transferencia, como GO TO, PERFORM, en la sección de acciones de la tabla controladora (nivel superior) dirige la ruta hacia tabla menores. Existen dos tipos de transferencia: directa y temporal.

IV.4.4.1. Transferencia directa.

La transferencia directa utiliza una transferencia de una sola vez: la tabla a la que se hace referencia no envía de regreso a la tabla general original. La proposición de acción "GO TO nombre de la tabla" indica qué tabla hay que examinar a continuación.

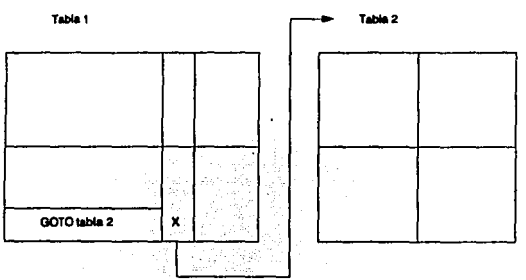


Fig. IV.7 Unión Múltiple de tablas utilizando la forma "GO TO"

IV.4.4.2. Transferencia Temporal.

En contraste, se une la tabla uno con la tabla dos por medio de la instrucción "PERFORM" tabla2." Al final de la tabla dos, una la instrucción RETURN guía el control de regreso a la instrucción que sigue a la de GO TO en la tabla uno.

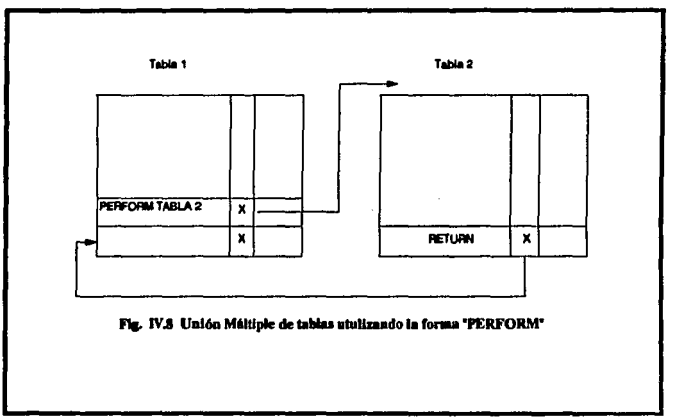


Fig. IV.8 Unión Múltiple de tablas utilizando la forma "PERFORM"

IV.4.5. Procesadores de tablas de decisión.

Las tablas de decisión se han automatizado parcialmente. Los procesadores de las tablas son programas de computadoras que manejan las formulaciones actuales de las tablas, con base en las entradas proporcionadas por el analista. También hacen toda la verificación de la redundancia y de la consistencia, algunas convierten el conjunto de decisiones y condiciones en instrucciones reales de programa de computadora.

La utilidad de los procesadores de tablas de decisión radica en el ahorro del tiempo de programación y verificación de errores.

IV.5. ESPAÑOL ESTRUCTURADO.

El español estructurado es un método adicional para superar los problemas de lenguaje ambiguo, al establecer condiciones y acciones en las decisiones y procedimientos. Este método no utiliza árboles o tablas, sino instrucciones descriptivas para definir un procedimiento. No muestra reglas de decisión: las establece.

Las especificaciones del español estructurado requiere que los analistas identifiquen las condiciones que ocurren dentro de un proceso, las decisiones que deben realizarse cuando se dan y las acciones alternas que hay que tomar. Sin embargo, este método también les permite enumerar los pasos en el orden en el que se deben efectuar. No se utilizan símbolos o formatos especiales, característica que a algunos les disgusta en el uso de los árboles de decisión y las tablas; también se pueden establecer completas con rapidez, dado que sólo se utilizan instrucciones parecidas al español.

La terminología utilizada en la descripción estructurada de una aplicación consiste en gran parte en nombres de datos para elementos que se almacenan en el diccionario de datos desarrollados para el proyecto.

IV.5.1. Desarrollo de Instrucciones Estructuradas.

El español estructurado utiliza tres tipos básicos de instrucciones para describir un proceso que trabajan bien para los análisis de decisión y pueden introducirse en la programación y desarrollo de software. Las tres son estructuras de secuencia, decisión e iteración.

IV.5.1.1. Estructuras de secuencia.

Una estructura de secuencia es un solo paso o acción incluida dentro de un proceso. No depende de la existencia de ninguna condición y, cuando se encuentra siempre se lleva a cabo. En forma normal se utilizan varias instrucciones de secuencia de manera conjunta para describir un proceso.

Por ejemplo, para comprar un libro probablemente se seguiría un procedimiento similar al que se describe a continuación:

1.- Escoger el libro deseado.

2.- Llevar el libro deseado al mostrador de salida.

3.- Pagar el libro.

4.- Recibir la nota.

5.- Salir de la tienda.

Este simple ejemplo muestra una secuencia de cinco pasos. Ninguno de éstos contiene una decisión para alguna condición que determine si se va a realizar o no; incluso se toman en el orden en el que se enumeran. En la compra del libro, por ejemplo, tendría poco sentido pagar por éste antes de escogerlo. Por lo tanto, el procedimiento se describe para señalar el orden correcto de las acciones.

IV.5.1.2. Estructuras de decisión.

El español estructurado es otra forma para mostrar el análisis de decisión; por lo tanto, las secuencias de acción descritas con frecuencia se incluyen dentro de las estructuras de decisión que identifican las condiciones. Las estructuras de decisión, por lo tanto, ocurren cuando dos o más acciones se pueden llevar a cabo, dependiendo del valor de una condición específica. Se debe considerar la condición y entonces tomar la decisión para cumplir con las acciones establecidas o conjuntos de acciones para esa condición. Una vez que se ha determinado que la condición se realiza, las acciones son incondicionales, como se mencionó anteriormente.

Se ampliará el ejemplo de la librería para mostrar la estructura de decisión. Cuando se entra a una librería, puede no encontrarse el libro buscado; por lo tanto, aquí se pueden mostrar las acciones para la condición: encontrar el libro deseado o no encontrar el libro deseado:

SI se encuentran los libros deseados, **ENTONCES**

Llevarlos al mostrador de salida.

Pagarlos.

Asegurarse de obtener su nota.

Salir de la tienda.

EN OTRO CASO

No llevarse nada.

Salir de la tienda.

Esta estructura de decisión, a través del uso de frases **SI/ENTONCES/EN OTRO CASO,** señalan las alternativas en èl proceso de decisión con bastante claridad. Se indican dos condiciones y dos acciones.

En el español estructurado se pueden incluir estructuras individuales dentro de otras estructuras. Escribir los detalles en un formato escalonado ayuda también a agrupar las condiciones y acciones que van juntas. La claridad extra que se gana a través de esta convención es más útil aun cuando se tiene que ver con situaciones de decisión grandes o complicadas. Las estructuras de decisión no se limitan a combinaciones de dos condiciones o acciones. Pueden existir muchas condiciones.

IV.5.1.3. Estructuras de iteración.

En las actividades de operación de rutinas es común encontrar que ciertas actividades se repiten mientras existe una cierta condición o hasta que ocurre cierta condición. Las instrucciones de iteración permiten a los analistas describir estos casos.

Al buscar un libro se pueden repetir los siguientes pasos:

MIENTRAS Todavía existen libros para examinar

Leer el título del libro

SI el título suena interesante

ENTONCES escoja el libro y hojéelo

Mire el precio

SI usted decide que desea el libro

Póngalo en el librero de **LIBRO ESTABLE**

SINO regréselo al librero

FIN DEL SI

SINO continúe

FIN DEL MIENTRAS

SI se encuentra los libros deseados **ENTONCES**

Lleve los libros al mostrador de salida

Pague los libros

Asegúrese de obtener su nota

Salga de la tienda

SINO

No lleve nada

Salga de la tienda

FIN DEL SI

En este ejemplo, se ve que cero o más iteraciones de la búsqueda de los libros deseados se están describiendo. Pasos adicionales se conjugan dentro de la curva de iteración. Proporcionan instrucciones sobre qué hacer cuando los títulos son interesantes y cuando se encuentra el libro deseado. La repetición continúa, mientras exista la condición de que todavía hay más libros que examinar, Entonces, el procedimiento muestra los pasos que han de llevarse a cabo y se han encontrado algunos libros deseados.

IV.5.2. Beneficios del Español Estructurado.

Se puede ver cómo el español estructurado puede ser útil para describir las condiciones y las acciones con claridad. Cuando se examina una actividad de negocios, los analistas deben utilizar el español estructurado para establecer qué acción tomar cuando se lleva a cabo una decisión, significa que necesitan adquirir más información para describirla. Por otro lado, después de que las actividades se han descrito en esta forma estructurada, los analistas pueden pedirle a otras personas que revisen la descripción y determinación con rapidez si se cometieron errores u omisiones en el establecimiento de los procesos de decisión.

CAPITULO V.

DISEÑO ESTRUCTURADO.

INTRODUCCION.

Diseñar, en informática puede entenderse como especificar las características de un sistema para que efectúe una función de la mejor manera posible según criterios técnicos.

Tradicionalmente, el único criterio de diseño ha sido la eficiencia, pero en un sentido limitado, solo considerando el uso de memoria y tiempo de proceso. La eficiencia seguirá siendo objetivo de diseño pero en el sentido más amplio del uso eficiente de los recursos más escasos del sistema. Estos recursos, además de la memoria central y el tiempo de proceso, incluyen: almacenamiento secundario, tiempo de periféricos de entrada y salida, líneas de teleproceso, tiempo de personal.

Por lo tanto el diseño estructurado es el arte de diseñar los componentes (o subsistemas) de un sistema y las interrelaciones entre estos componentes, de la mejor manera posible.

El diseño estructurado, no crea una nueva actividad en el proceso de desarrollo de los sistemas, simplemente consolida, formaliza, hace visible actividades y decisiones que ocurren inevitablemente durante el desarrollo de un proyecto. Las decisiones de diseño se pueden abordar concientemente como alternativas evaluables, en vez de proceder a tientas adivinando, por suerte o por omisión.

En la construcción de un sistema, el diseño estructurado determina cuántas partes habrá, que hace cada una, cómo están organizadas y cuales son las interrelaciones entre ellas para resolver el problema en cuestión.

V.1.- El DIAGRAMA DE ESTRUCTURA.

El diagrama de estructura es una representación gráfica del sistema, que se utiliza como técnica para el diseño, implantación, documentación y mantenimiento del sistema. Es un modelo independiente del tiempo, de las relaciones entre los módulos, de un programa o de un sistema. Es por ésto que de un diagrama de estructura no se puede inferir cual es el orden en el que se ejecutan los módulos.

Los elementos que forman al diagrama de estructura son los siguientes:

- Módulos
- Conexiones
- Interfaces

En la figura V.1 se muestra la notación de los diagramas de estructura.

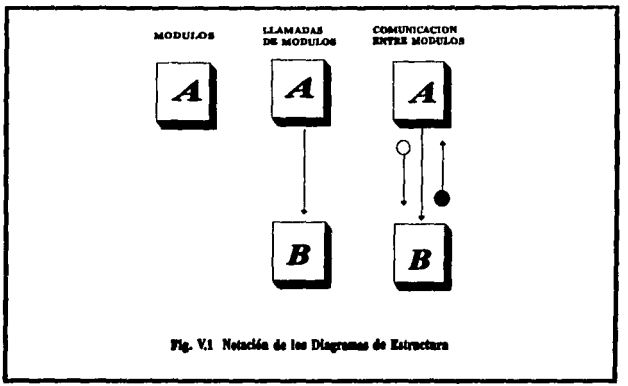


Fig. V.1 Notación de los Diagramas de Estructura

V.1.1. El módulo.

Un módulo es una secuencia de instrucciones continuas de un programa limitadas por elementos frontera, que tienen asociado un identificador. por ejemplo:

- En PL/1: "Procedure"
- En COBOL: Programa, subprograma, "Section", "Paragraph"
- En FORTRAN : "Subroutine", "Function"

Los módulos tienen los siguientes atributos:

El qué :

- Entradas (lo que obtiene de su invocador).
- Salidas (lo que regresa a su invocador).
- Función (Lo que hace a las entradas para producir las salidas).

El Cómo :

- Mecánica (Como hace su función).
- Datos Internos (Espacio privado de trabajo, solo él los referencia).

Además tienen un nombre con el cual son identificados, dentro del todo y pueden ser llamados por otros módulos, por ejemplo, usando la declarativa "CALL" en FORTRAN.

El nombre de un módulo debe definirse como una declaración imperativa que describa la función que realizará.

En la figura V.2 se muestra la representación gráfica de un módulo.

Lógicamente no hay distinción entre un módulo, un programa y un sistema. Pero físicamente las diferencias se dan de programas grandes; en cuanto a operación de sistemas se puede hablar de un programa pero usualmente no de los módulos del programa.

Fig. V.2. Representación gráfica de un módulo

V.1.2 Conexiones.

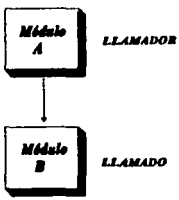
Las conexiones o referencias a elementos o a identificadores dentro de un programa pueden ser de dos tipos: normales y patológicas. Es una normal si está es a la interfase de la identificación del módulo. Cuando la referencia entre módulos es un identificador en el interior de otro, la conexión se llama patológica.

En la figura V.3 se muestra la representación de las llamadas de los módulos.

V.1.3 Interfases.

La interfase es la porción de los límites de un sistema o de parte del mismo, por la que pasan entradas o salidas. La interfase puede ser de datos o de control, dependiendo de la naturaleza del flujo de la conexión.

En la figura V.4. se muestra la representación de la comunicación entre módulos. Así pues, el módulo OBTENER DETALLES DEL LECTOR no solo llama a ENCONTRAR NOMBRE DEL LECTOR:



ESTO SIGNIFICA :

A LLAMA A -> B

B REALIZA SU FUNCION

B REGRESA EL CONTROL A "A" INMEDIATAMENTE DESPUES DEL PUNTO DONDE FUE LLAMADO

Fig. V.3. Representación de las llamadas entre módulos

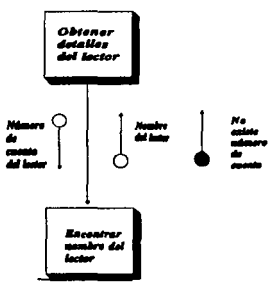


Fig. V.4. Representación de la comunicación entre módulos.

- OBTENER DETALLES DEL LECTOR envía el número de credencial del lector para encontrar el nombre del mismo.
- ENCONTRAR NOMBRE DEL LECTOR regresa el nombre a OBTENER DETALLES DEL LECTOR.
- ENCONTRAR NOMBRE DEL LECTOR regresa la información de control NO EXISTE NUMERO DE CREDENCIAL a OBTENER DETALLES DEL LECTOR.

En la figura V.5 se muestra el significado de las interfases.

En la figura V.6 se muestra un ejemplo de diagrama de estructura.

Como se puede observar ENCONTRAR DETALLE DE LIBROS aparece solo una vez pero es llamado por dos módulos que se le conoce como principales.

El llamado de módulos no se hace necesariamente de izquierda a derecha.

No se puede saber cuantas veces OBTENER REGISTRO DE PRESTAMOS DEL LECTOR llama a ENCONTRAR REGISTRO DE PRESTAMOS.

El nombre del módulo más alto es la función. En este ejemplo es APROBACION DE LIBROS Y LECTOR.

La diferencia entre un diagrama de bloque y un diagrama de estructura se muestra en la figura V.7.

INTERFASE	SIGNIFICADO
●——→	EL CIRCULO INDICA EL TRANSMISOR Y LA FLECHA AL RECEPTOR DE LA INFORMACION
●——→	DATOS QUE VIAJAN EN LA INTERFASE ES LLAMADO UN ACOPLAMIENTO DE DATOS
○——→	INFORMACION DE CONTROL: INDICA SI EL SUBORDINADO TUVO EXITO EN EL TRABAJO ENCOMENDADO, ES LLAMADO UN ACOPLAMIENTO DE CONTROL.

Fig. V5. Significado de las interfases entre módulos.

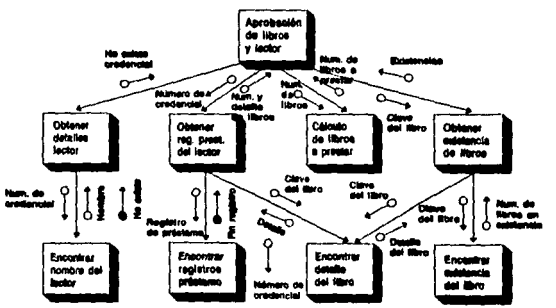


Fig. V.6. Ejemplo de un diagrama de estructura

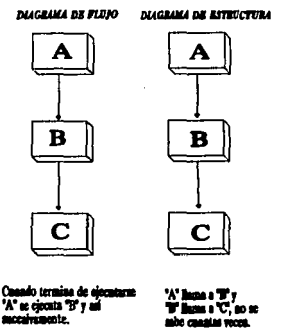


Fig. V.7. Diferencia entre diagramas.

Un diagrama de estructura muestra lo siguiente:

- La división del sistema en módulos.
- Jerarquía y organización de los módulos.
- Interfases de comunicación entre módulos.
- Nombre (funciones de los módulos).

El diagrama de estructura no muestra mecanismos internos de los módulos.

En la figura V.8 se muestra el diagrama de estructura para el mantenimiento de archivos.

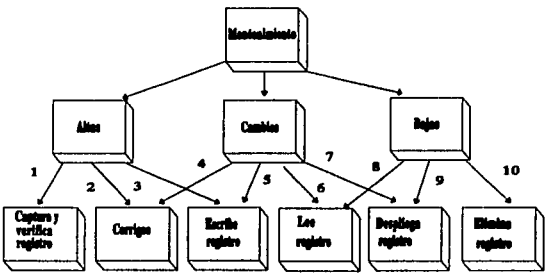


Fig. V.8 Diagrama de estructura de mantenimiento de archivos.

V.1.4. Métodos de Especificación de Módulos.

En esta parte se tratará como sirve el diagrama de estructura a los programadores para que deriven el código.

Esto se hace por medio de las especificaciones de cada módulo en el diagrama de estructura.

Hay tres métodos para especificar un módulo :

- Especificación de las interfases de los módulos.
- Especificación por medio de técnicas de análisis
- Especificación por lenguaje Estructurado o Pseudocódigo.

El especificar los módulos a los programadores es tocar algunos aspectos de las políticas de la organización, ya que pueden surgir conflictos cuando al programador se le indica como realizar su trabajo.

V.1.4.1. Especificación de Interfases entre módulos.

La especificación de interfases entre módulos proporciona una cantidad mínima de detalles acompañando al diagrama de estructura, y se pueden incluir algunas descripciones físicas como por ejemplo "PIC 9(03)" en COBOL. Se debe decir al programador que entradas se proporcionan al módulo cuando es llamado, que salidas se obtienen de éste cuando retorna y la función que el mismo debe realizar.

En la figura V.9 se muestra la especificación de las interfases entre módulos.

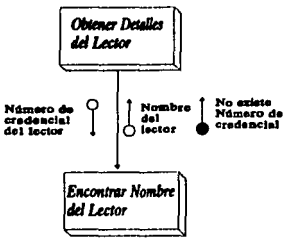


Fig. V.9 Especificación de las interfases entre módulos

A continuación se describen las especificaciones de las interfases para el módulo ENCONTRAR NOMBRE DEL LECTOR:

Módulo	Encontrar nombre del lector
Función	Encontrar el nombre del lector
Usa	Número de credencial del lector PIC 9(05)
Regresa	El nombre del lector PIC X(30)
	Bandera de control PIC X(01).

V.1.4.2. Especificaciones Usando Técnicas del Análisis.

Como se mencionó en el capitulo IV, existen tres métodos para el análisis de decisiones que son: los árboles de decisión, las tablas de decisión y el lenguaje estructurado.

V.1.4.3. Especificaciones por Lenguaje Estructurado o Pseudocódigo.

El pseudocódigo es la forma más detallada de describir un módulo. Es un lenguaje intermedio entre el lenguaje natural y el de programación en que se intenta implantar la solución. Existen dos diferencias entre el pseudocódigo y el lenguaje estructurado:

- El pseudocódigo no es una técnica de usuarios y analistas, es más bien de diseñadores y programadores.
- El pseudocódigo no debe ser visto por los usuarios porque no están familiarizados con éste.

A continuación se detalla un módulo, éste determina si procede el prestamo de libros a un lector.

```
Si NUMERO DE CREDENCIAL no existe
O si NUMERO DE LIBROS es menor o igual a NUMERO DE LIBROS
MINIMO
O si LECTOR ADEUDA LIBROS
Entonces
Mueve "NO" a PROCEDE - PRESTAMO
De otra forma
Mueve "SI" a PROCEDE - PRESTAMO
Fin de si
```

V.1.5. Comunicación entre diseñador y programador.

La técnica que el diseñador utilice para especificar los módulos a los programadores no debe suplantar a la comunicación tradicional, ya que, si se crean barreras entre las personas, se obtendrán malestares. No hay que separar al equipo de programación del de diseño. Se debe permitir a los programadores el acceso al diagrama estructurado antes de que éste se ha terminado, de lo contrario habrá confusiones y la técnica de especificación utilizada no será buena.

V.1.6. Cualidades de un buen diseño.

Uno de los principios fundamentales de diseño es que un sistema grande se puede dividir en módulos manejables. Asimismo es vital que el particionamiento se realice de tal forma que los módulos sean lo más independiente posible y que cada uno se ejecute solo, es decir, que su función únicamente sea la relacionada con su problema. Al primer criterio se le llama acoplamiento modular mientras que al segundo se le conoce como cohesión o fuerza modular.

Además de los conceptos de particionamiento, acoplamiento y cohesión, que son el tema principal de éste capítulo, hay muchas otras guías que se pueden usar para evaluar y mejorar la calidad de un diseño.

V.1.6.1. Acoplamiento Modular.

Como se ha expuesto anteriormente, al dividir un problema en partes independientes, se simplifica tanto la comprensión como el desarrollo de es parte. Para medir la independencia entre módulos existen criterios que permiten evaluarla.

Un módulo es independiente si puede funcionar sin la presencia de algún otro. No es lógico que exista un módulo que cumpla totalmente con esa característica pues lo normal es que reciba información o produzca algún resultado. Por lo tanto, siempre debe haber un interfase por la cual se le pase el control y los datos y además regrese resultados. La complejidad de dicha interfase depende de dos factores:

- La cantidad de información que se pasa.
- La disponibilidad de la información.
- La estructura de la información.

Estas interfases deben mantenerse simples con un número reducido de parámetros, de manera que el programador pueda encontrar fácilmente la referencia.

Su estructura debe de ser obvia, nunca obscura.

Para ser módulos independientes es necesario reducir el acoplamiento entre los mismos. El acoplamiento es una medida del grado de independencia entre módulos al programar.

Desde el punto de vista operativo, es la probabilidad de que al programar, corregir o modificar un módulo, sea necesario tomar en cuenta elementos de otro.

Existen cuatro factores que afectan el acoplamiento:

- El tipo de acoplamiento. El acoplamiento menor lo hacen las conexiones mínimas siendo mayor el acoplamiento en las conexiones normales y muy alto en las condiciones patológicas.
- Complejidad de la interface. Número de parámetros que se pasan.
- Tipo de información que fluye por la conexión. El flujo de datos produce menor acoplamiento que las conexiones híbridas.
- Tiempo en que ocurre la conexión. La conexión a tiempo de ejecución tiene

menor acoplamiento que las de tiempo de conexión (LINKAGE), menor que el que ocurre al tiempo de compilación y menor que al tiempo que de codificación, en ese orden.

Los diferentes tipos de acoplamiento son:

Tipos de Acoplamiento	Nivel de Acoplamiento
Acoplamiento de datos	Mejor
Acoplamiento estampado	
Acoplamiento de control	
Acoplamiento de área común	
Acoplamiento de contenido	
Acoplamiento Híbrido	Peor

V.1.6.2. Acoplamiento de Datos.

Ocurre cuando solamente se pasan datos con información entre los módulos, sin elementos de control como indicadores, banderas, etc.

Este tipo de acoplamiento es el más deseable, y de hecho cualquier sistema puede construirse de esa manera. Un ejemplo de este acoplamiento se muestra en la figura V.10.

Hay que tener mucho cuidado con los datos que viajan a lo largo del sistema sin que sean usados por la mayoría de los módulos por los que pasan, son únicamente un puente como se muestra en la figura V.11.

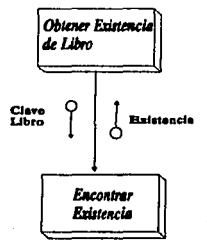


Fig. V.10 Acoplamiento de Datos.

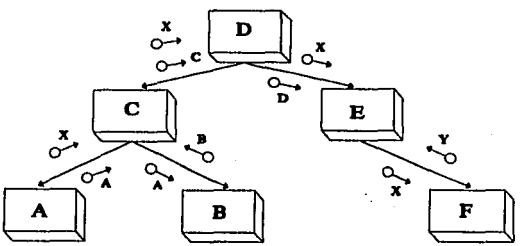


Fig. V.11 Datos Puente entre Módulos

V.1.6.3. Acoplamiento Estampado.

Dos módulos presentan acoplamiento estampado si hacen referencia a la misma estructura de datos (no global).

Una estructura de datos esta compuesta de elementos de datos, por ejemplo, una tabla, un registro, etc.

Este tipo de acoplamiento presenta el siguiente problema: Un cambio en la estructura de datos afectará a todos los módulos que estan "Estampados" en la misma, sin embargo, usando una adecuada estructura de datos, este tipo de acoplamiento se acerca al de datos. Un ejemplo es mostrado en la figura V.12.

El problema aumenta cuando un módulo pasa el total de la estructura pero solamente necesita parte del registro como en el ejemplo anterior solo se necesita la clave del libro y si hay un cambio en el registro del mismo, tendrá que afectar el módulo de ENCONTRAR EXISTENCIA.

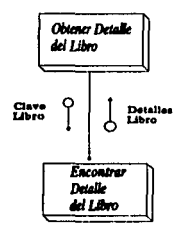


Fig. V.12 Acoplamiento Estampado.

V.1.6.4. Acoplamiento de Control.

Dos módulos presentan acoplamiento de control si se comunican usando al menos un elemento de control (bandera, switch, indicador).

Este acoplamiento es indeseable porque uno o ambos módulos no serán una caja negra, es decir, el proceso que este realiza es complejo y es necesario tener conocimiento del mismo. Esto se muestra en la figura V.13.

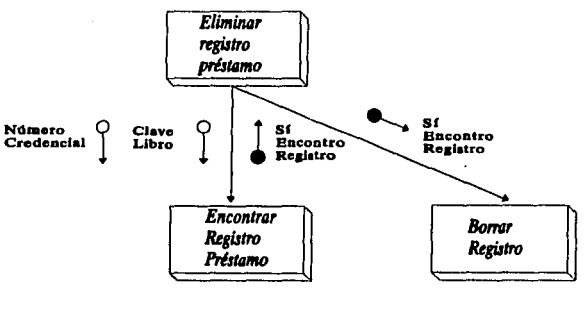


Fig. V.13 Acoplamiento de Control

V.1.6.5. Acoplamiento de Area común.

Un grupo de módulos presentan este tipo de acoplamiento si comparten una misma área global de datos.

Surgen varios problemas cuando se presenta lo anterior:

- Es más difícil reusar los módulos que utilizar áreas comunes ya que usualmente lo hacen por su nombre.
- El mantenimiento se hace más difícil sobre todo cuando se pasan diferentes tipos de datos a través del área común.
- El uso de áreas globales dificulta la legibilidad de los problemas.
- Es difícil saber que módulos usan que datos.
- Un error en cualquier módulo usando el Area común puede aparecer en otro (que también use esa área) ya que esta no se encuentra protegida por ningún módulo.

V.1.6.6. Acoplamiento de contenido.

Ocurre cuando:

- Un módulo altera instrucciones en otro.
- Un módulo hace referencia o cambia de datos contenidos en otro.
- Un módulo brinca al otro.
- Otros módulos comparten las mismas literales.

Los tres primeros puntos se conocen como patológicos.

El problema principal es que un cambio pequeño y de apariencia inocente a uno de los módulos puede afectar a otro en cualquier parte del sistema, como se muestra en la figura V.14.

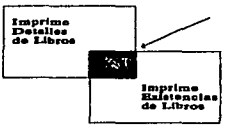


Fig. V.14 Acoplamiento de Contenido

V.1.6.7. Acoplamiento Híbrido.

El Acoplamiento híbrido es cuando existen dos o más significados para el mismo campo de datos. Este problema se puede observar con la declarativa "REDEFINES" de COBOL. Este es el peor de todos los acoplamientos.

V.1.7. Cohesión o Fuerza Modular.

Este concepto se refiere a que tan relacionados estan los elementos internos del módulos con respecto a la función que este realiza. Existe una gran relación entre la cohesión y el acoplamiento. A mayor cohesión de un módulo, menor será el acoplamiento con otros. En la fase de diseño es más fácil prever la cohesión de un módulo antes de construirlo, que su acoplamiento. Este es más fácil de ver en un sistema ya construido y sirve para analizar las estructuras diseñadas con el fin de mejorarlas.

Para analizar la cohesividad de un módulo se debe tomar en cuenta todos los procesos incluyendo el que se hace en los módulos subordinados que son llamados por éste.

Los diferentes niveles de cohesión de mayor a menor son los siguientes:

Tipos de Cohesión	Nivel de Cohesión
Cohesión Funcional	Mejor
Cohesión secuencial	
Cohesión comunicacional	
Cohesión de procedimiento	
Cohesión temporal	
Cohesión Lógica	
Cohesión concidental	Peor

V.1.7.1. Cohesión Funcional.

Se da cuando todos los elementos del proceso son parte integral y esencial para realizar una sola función.

V.1.7.2. Cohesión Secuencial.

Sucede cuando los datos de salida de un elemento del proceso sirven de entrada al

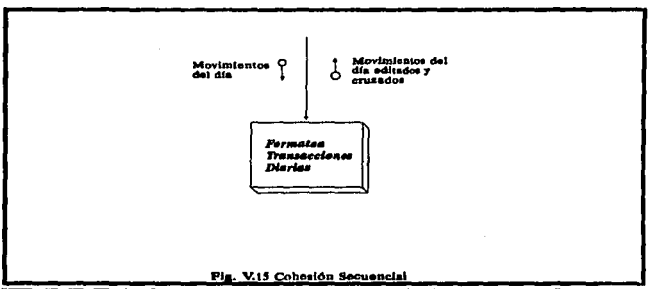


Fig. V.15 Cohesión Secuencial

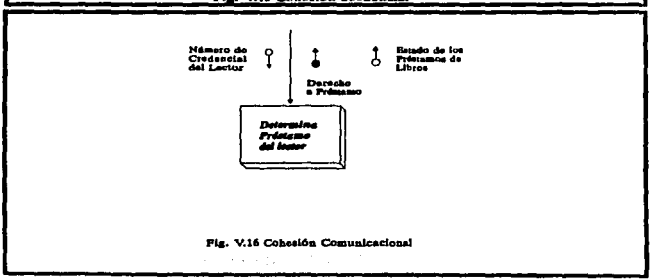


Fig. V.16 Cohesión Comunicacional

siguiente. Este tipo de cohesión generalmente resulta de diseñar los módulos a partir de diagramas de bloque. Un módulo secuencial puede contener la totalidad de las funciones o solo parte de alguna como se indica en la figura V.15.

V.1.7.3. Cohesión Comunicacional.

Un módulo con ese tipo de cohesión es aquel en el que sus elementos contribuyen a diferentes tareas, pero cada una se refiere a los mismos parámetros de entrada y/o salida, como lo muestra la figura V.16.

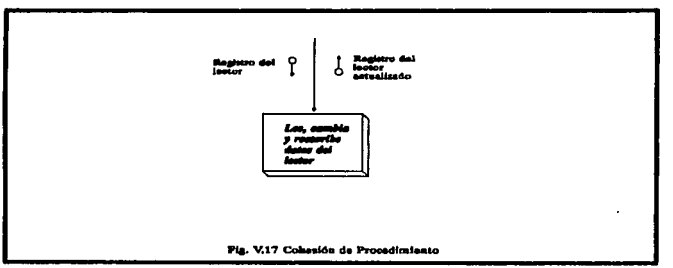


Fig. V.17 Cohesión de Procedimiento

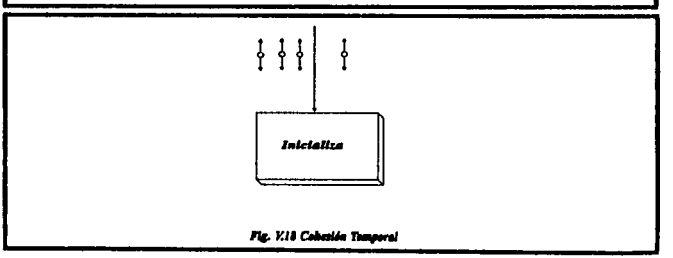


Fig. V.18 Cohesión Temporal

V.1.7.4. Cohesión por Procedimiento.

Ocurre cuando se agrupan elementos en un proceso por pertenecer a un mismo procedimiento, es decir, cuando el control fluye de un elemento al siguiente, pero los datos no necesariamente lo hacen de la misma manera como se ilustra en la figura V.17.

V.1.7.5. Cohesión Temporal.

Se da cuando los elementos se relacionan por el tiempo y no por la función determinada como se presenta en la figura V.18.

V.1.7.6. Cohesión Lógica.

Ocurre cuando los elementos de un módulo se clasifican como relacionados a una misma clase de funciones, un ejemplo se puede observar en la figura V.19.

V.1.7.7. Cohesión Concidental.

Ocurre cuando no hay ninguna o existe poca relación entre los elementos de un módulo. Solo estan juntos por casualidad y generalmente resultan de dividir en módulos un programa ya codificado o cuando se trata de ahorrar memoria. También se da cuando al codificar un programa para el que no se aplicaron técnicas de diseño estructurado, se trata de evitar codificación repetida, como se muestra en la figura V.20.

Los tres primeros tipos de cohesión son los más aceptables, los demás producen diseños con un alto costo de desarrollo y mantenimiento.

Para indentificar el tipo de cohesión de un módulo, se hace una prueba lingüística. Si la función de un módulos se puede describir con una oración imperativa con un solo verbo

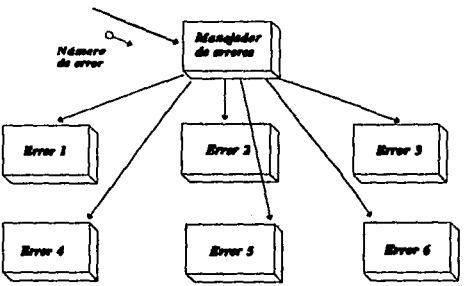


Fig. V.19 Cohesión Lógica

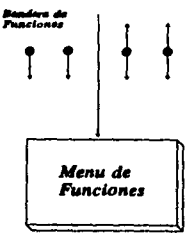


Fig. V.20 Cohesión de Funciones

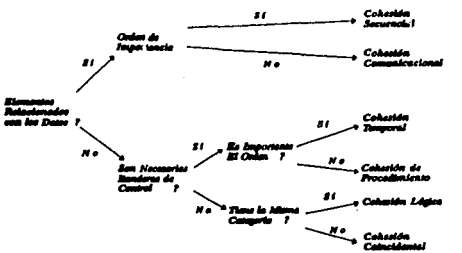


Fig. V.21 Tipos de Cohesión Interna de los Módulos

transitivo y un sujeto en singular, el módulo será funcional.

Los módulos lógicos tienen en la oración que los describe el sujeto en plural e incluye la palabra todos. Si la oración contiene palabras que impliquen tiempo o secuencia, una coma o más de un verbo, el módulo descrito puede ser temporal, por procedimiento o comunicacional. En la figura V.21 se muestra un resumen de la cohesión interna de los módulos.

V.2. ENTRE EL ANALISIS Y EL DISEÑO.

Una vez definidos los principales conceptos y técnicas que se deben manejar en la fase de diseño, es necesario explicar la forma en la que los diagramas de flujo de datos se toman como base para que a través de un proceso llamado empaquetamiento del análisis, se obtenga el diagrama de estructura del sistema.

Uno de los principios de todas las disciplinas estructuradas señala que cada medio particular de realizar algo debería posponerse tanto como sea posible. De esa forma, primero se determinan los requerimientos antes de escoger un medio particular de llevarlos acabo.

El empaquetamiento es un conjunto de decisiones y actividades que subdividen a un sistema en unidades específicas para llevarlas acabo. Dichas unidades pueden ser de diferentes formas y tamaños como por ejemplo: un sistema, un trabajo, un paso de trabajo, un programa, una unidad de carga, etc.

Para entender lo anterior se definen algunos términos:

Trabajo es una secuencia de uno o más pasos.

Paso de trabajo es un programa principal con (opcionalmente) una jerarquía de uno o más subprogramas. Un programa principal es llamado por el sistema operativo, un subprograma es llamado por otro programa.

Programa es una jerarquía de uno o más módulos de un diagrama de estructura.

Una Unidad de carga es una jerarquía de uno o más programas cuya llamadas han sido encadenadas antes de su ejecución.

Al finalizar la fase de análisis, se debe empacar el sistema en trabajos separados y en pasos de trabajo.

V.2.1. Empaquetado en trabajos separados.

El empaquetar un sistema en varios trabajos se realiza al final del análisis por varias razones:

- Porque se requiere alguna información del usuario. La misma esta disponible después que el análisis ha sido terminado.
- Los diagramas de flujos de datos desarrollados mediante el análisis son útiles para mostrar en donde cruzan fronteras los flujos de datos.
- Facilita el trabajo del diseñador.
- Poder observar al sistema como una red de trabajos distintos es más útil que tratar con el sistema como un trabajo enorme.

Para empacar el sistema en trabajos se deben dibujar las siguientes tres fronteras:

- Frontera de hardware. Diferentes partes del sistema pueden ser implantadas en diferentes máquinas.
- Fronteras de proceso por lote/en línea. Se deben determinar que partes del sistema realizarán su proceso en línea, cuales en lote y cuales en tiempo real.
- Fronteras de ciclos (alias periodicidad). Se deben empaquetar en forma separada los procesos de trabajos con diferentes ciclos de tiempo o periodicidad. Asímismo, los procesos de trabajo con el mismo tiempo de ciclo también deben separarse cuando la única relación entre ellos solo sea temporal.

Al empacar el sistema en trabajos, se deberán hacer ciertas preguntas cada vez que un flujo de datos cruce cualquier frontera física. Esto sirve para determinar entre otras cosas, los costos de implantación y ejecución del sistema:

- Qué volumen de datos cruza la frontera ?
- Cuál es la frecuencia probable de los datos ?
- Tendrá que producirse alguna salida dentro de un tiempo crítico después que se ha recibido la entrada ?
- En qué dispositivo será capturado/desplegado un dato ?
- Tienen que introducirse algunos archivos extra ?
- Se necesita algún método especial de acceso de datos ?
- Qué tan fácil es para diferentes trabajos, compartir archivos ?
- Qué protocolo máquina / máquina se requiere ?
- Que requerimientos de validación (en la entrada) existen ?
- Que requerimientos de formato hay sobre la entrada o sobre la salida ?
- Cuales usuarios serán afectados, y cómo ?
- Qué tan valiosos son los datos para la organización ?
- Qué aspectos de seguridad se debe realizar sobre los datos?

V.2.2 Empaquetamiento en pasos de trabajo.

El fragmentar un trabajo en pasos se realiza al final del análisis mediante el diagrama de flujo de datos. La regla para crear los pasos de trabajo es: Fragmentar el diagrama de flujo de datos en la menor cantidad posible de pasos de trabajo, con lo cual, disminuye el número de archivos intermedios necesarios y se reduce el tiempo de ejecución del sistema.

Se deben tomar en cuenta las siguientes consideraciones al dividir en pasos un trabajo:

- Paquetes de Software comercial. Estos paquetes pueden ser usados para ejecutar algunas de las funciones del sistema que deben hacerse para cada paso de trabajo.

- Los requerimientos de seguridad. Se puede requerir la introducción de pasos de trabajo y/o archivos intermedios que de otra manera serian innecesarios. Por ejemplo, requerimientos de auditoría, seguridad, respaldo y recuperación.

- Los recursos pueden ser insuficientes para ejecutar todo el trabajo a la vez. Si un proceso completo no se puede mantener en la máquina a un tiempo, entonces tendrá que dividirse en pequeños segmentos que puedan ser ejecutados uno a la vez.

V.3. ESTRATEGIAS DE DISEÑO.

Existen varias estrategias de diseño, a continuación se describen las más importantes.

V.3.1 El análisis de transformación.

El análisis de transformación es una estrategia de diseño cuyo objetivo es el derivar una jerarquía modular inicial en base al diagrama de flujo de datos de las funciones a ser desarrolladas. Para llevar a cabo tal estrategia, es necesario explicar algunos problemas que pueden presentarse y la forma de atacarlos.

V.3.1.1 Problemas que se presentan.

La estrategia de análisis de transformación puede aplicarse al diagrama de flujo de datos de la figura V.22 para producir el diagrama de estructura equivalente de la figura V.23. Sin embargo, la mayoría de los diagramas de flujo de datos tienen marcadas diferencias entre sí:

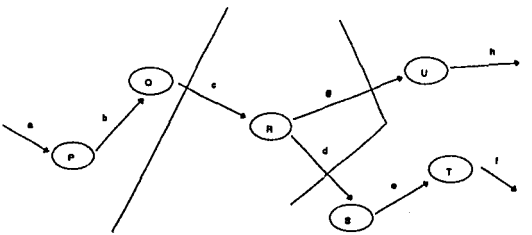


Fig. V.22 Diagrama de Flujo de Datos

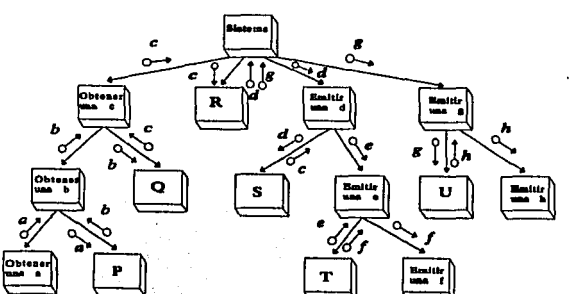


Fig. V.23 Diagrama de Estructura

- Los diagramas de flujo de datos son dibujados en niveles para permitir un enfoque descendente. La estrategia de transformación no dice como tratar con dichos diagramas.

- Además están construidos bajo el supuesto de que los datos fluyen hacia donde son requeridos. No se hace hincapié en la forma de obtener la entrada, emitir la salida, determinar tipo de datos o enviarlos hacia las funciones que los requieran, por lo que al aplicar la estrategia de diseño a éstos no se obtendrá un diagrama de estructura manejable.

- Dado que las banderas y switchs de control no se toman en cuenta en los diagramas de flujo de datos, el diagrama de estructura resultante puede requerir la adición de algunos acopladores de control que le permitan sincronizarse por sí mismo.

- Igualmente tienden a ser no lineales. Dado que la estrategia de análisis de transformación está orientada a dar mayor apoyo a los que si son lineales, el diagrama de estructura resultante a menudo no es el que se espera, donde todo o la mayor parte del diagrama de flujo de datos se juzga que no es aferente ni eferente, la transformación puede producir únicamente un diagrama tan simple como entrada-proceso-salida.

V.3.1.2 Extensiones requeridas a la estrategia de transformación.

Con el objeto de atacar los problemas antes expuestos son necesarios los siguientes pasos previos al proceso de transformación:

- Adicionar extremidades físicas. Adicionar a los diagramas de flujo de datos burbujas que explícitamente obtengan y emitan datos de sus contenedores físicos, es decir, añadir burbujas en el lado aferente para determinar el tipo de los datos y los flujos que llegan.

- Hacer los diagramas de flujo de datos en forma lineal, tanto como sea posible expresando las burbujas a nivel primitivo en la medida en que éstas permanezcan visiblemente aferentes, eferentes y lineales.

- Amalgamar la porción central del diagrama del flujo de datos en una sola burbuja de nivel superior.

Con los puntos antes tratados, el diagrama de estructura será manejable y tendrá una correcta elección del modelo principal, pero requerirá alguna factorización adicional de la porción central, para ésto se debe hacer:

- Factorizar la porción central siguiendo las mismas decisiones de fragmentación que se hicieron en los diagramas de flujo de datos por niveles, es decir, asignar un módulo de nivel **n** por cada burbuja correspondiente de nivel **n**. Por cada flujo de datos que llegue, se debe añadir un acoplador correspondiente enviado por el módulo principal.

- Por cada flujo que parta de la burbuja se debe añadir un correspondiente acoplador que se envia el módulo principal.

V.3.1.3 Estrategia modificada del análisis de transformación.

A continuación se describen los pasos necesarios para llevar a cabo la estrategia de análisis de transformación:

- **Paso 1** Adicionar al diagrama de flujo de datos extremidades físicas (que traten con datos de entrada/salida en sus contenedores físicos).

- **Paso 2** Linealizar (colocar en forma de linea) el nivel de primitiva funcional mediante la expresión de las burbujas aferentes y eferentes.

- **Paso 3** Seguir las principales corrientes aferentes hacia adentro (rumbo al centro) hasta que éstas alcancen el punto de más alta abstracción.

- **Paso 4** Efectuar el paso 3 para las corrientes eferentes.

- **Paso 5** Identificar la "transformación central" en el punto intermedio. Particionar todas las funciones contenidas en la transformación central de manera que todas ellas juntas sean representadas por una sola burbuja de nivel superior.

- **Paso 6** Dibujar un diagrama de estructura de nivel superior con un módulo principal y un subordinado por cada una de las corrientes aferente/eferente y un módulo subordinado para la transformación central.

- **Paso 7** Factorizar los módulos de las corrientes aferente y eferente hasta que se haya alcanzado la entrada física.

- **Paso 8** Factorizar el módulo de la transformación central, descomponiendo en forma paralela las porciones correspondientes del conjunto de diagramas de flujo de datos por niveles.

- **Paso 9** Adicionar acopladores de control (switches y banderas) para sincronizar la estructura.

- **Paso 10** Refinar el diagrama de estructura generado usando heurística de acoplamiento, cohesión y diseño.

A continuación se presenta un ejemplo de análisis de transformación.

Suponiendo que el diagrama de flujo de datos de nivel superior es como el de la figura V.24, la transformación procede de la siguiente forma:

Paso 1 Añadir extremidades físicas.

Se debe determinar la forma física más elemental de cada una de las corrientes de entrada y salida. En el caso de la figura V.24, suponer que la de entrada es una tarjeta y que las de salida son un registro en cinta y una línea impresa. Las extremidades físicas requeridas son una burbuja para transformar las tarjetas que entran en los flujos de datos lógicos a y b, y dos burbujas en el lado de salida para transformar los flujos de salida c, d y e en sus equivalentes físicos. El diagrama de flujo de datos resultante se muestra en la figura V.25.

Paso 2. Linealizar a lo largo de los flujos aferentes y eferentes.

Analizando los niveles más bajos de la transformación y suponiendo que U no es lineal, W es lineal y V es primitiva, se tiene que, trabajando a partir del exterior, se deben expresar las funciones a nivel primitivo en la medida en que permanezcan lineales. En este caso se

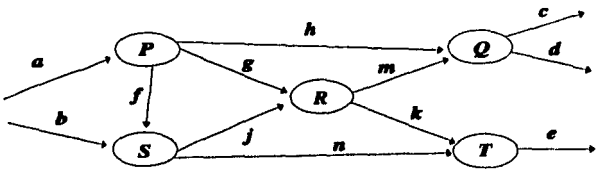


Fig. V.24 Diagrama de Flujo de Datos

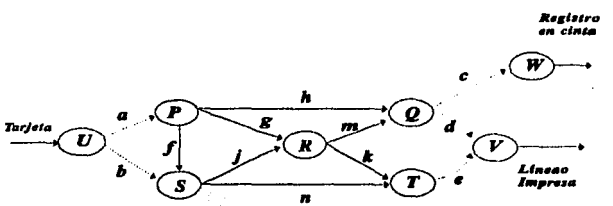


Fig. V.25 Añadir Extremidades Físicas

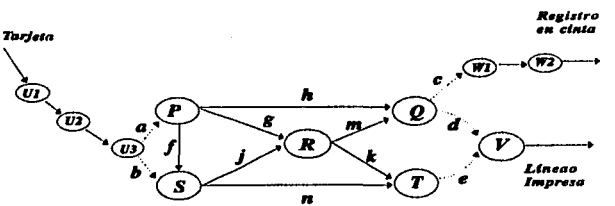


Fig. V.26 Linealizar a lo Largo de las Corrientes Aferentes y Eferentes

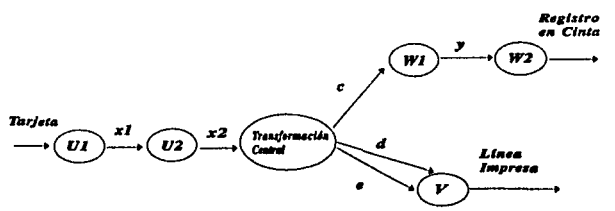


Fig. V.27 Identificar y Abstraer la Transformación Central

reemplaza la burbuja W por su real primitiva equivalente (W1 y W2). La burbuja V queda tal como está, mientras que la burbuja U se reemplaza por sus primitivas U1 y U2 (la porción lineal de la extremidad) y el resto representado por U3. En la figura V.26 se muestra el resultado obtenido.

Paso 3 y 4. Seguir corrientes aferentes y eferentes.

Debido a que a y b no pueden considerarse como un artículo obtenible, es probable que la corriente aferente termine en X2. En lo que respecta a las corrientes eferentes c,d y e son probablemente más lógicas.

Paso 5. Identificar y abstraer la transformación central.

La transformación central es la parte entre X2, c, d y e. El resultado de agrupar dicha porción es una burbuja se muestra en la figura V.27.

Paso 6. Dibujar el diagrama de estructura de nivel superior.

El módulo principal tendrá un nombre que identifique lo que es el todo o la estructura completa. El número de módulos subordinados son siempre uno o más que el número total de corrientes aferentes y eferentes. En el ejemplo, los módulos subordinados serían cuatro: la transformación central más la corriente de la tarjeta, la de la cinta y la de la impresora. El resultado se observa en la figura V.28.

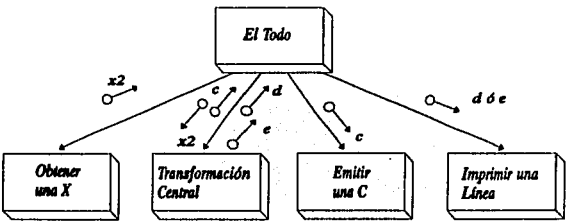


Fig. V.28 Diagrama de Estructura de Nivel Superior

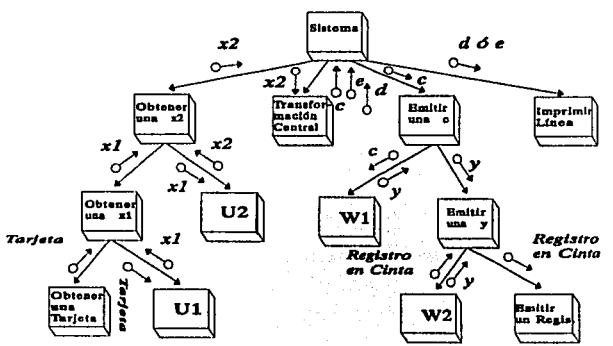


Fig. V.29 Diagrama de Estructura Resultante

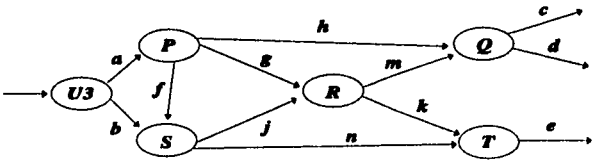


Fig. V.30 Hijo de la Burbuja de Transformación Central

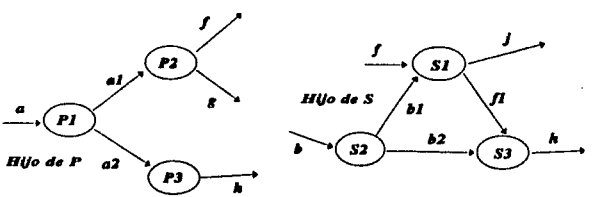


Fig. V.31 Burbujas Nietas de la Transformación Central

Procedimiento hacia el exterior y partiendo de la transformación central, habrá tantos niveles en la estructura como burbujas haya en la salida hacia el eje físico. Cada nivel se debe componer de una estructura binaria, que consiste de, en el lado aferente, un módulo que obtenga y otro que transforme. El acoplador es el mismo que el flujo de datos que se muestra entrando en la burbuja asociada en el diagrama de flujo de datos. Dicho acoplador pasa al transformador, el cual lo usa para construir el acoplador que es el que sale de la burbuja en el diagrama de flujo de datos. El módulo transformador puede usar el mismo nombre que el asignado a la burbuja asociada. El diagrama de estructura resultante se muestra en la figura V.29.

Paso 8. Factorizar el módulo de transformación central.

El hijo de la burbuja de transformación central se muestra en la figura V.30. Algunos ejemplos de figuras nietas se muestran en la figura V.31. Al adaptar la descomposición de dichas figuras de los niveles inferiores, se puede construir dos niveles adicionales debajo del módulo de transformación central en la gráfica de estructura. El resultado se muestra en la figura V.32.

Paso 9. Añadir acopladores de control para sincronizar la estructura.

Debido a que es imposible que los módulos realicen sus funciones sin la adición de algún control de acoplamiento, en general, todo módulo en la estructura derivada puede requerir de algunos acopladores de control con el objeto de realizar su trabajo.

Paso 10. Refinar usando acoplamiento, cohesión, etc.

El diagrama de estructura que se obtenga al aplicar los pasos 1 al 9 probablemente tenga algunas características totalmente inaceptables, como módulos poco cohesivos, sobreacoplamiento o violación de alguna o toda heurística del diseño. Es por ello que es necesario fragmentar más a los módulos, reagruparlos o moverlos de lugar en el diagrama de estructura con el objeto de maximizar su cohesión interna y minimizar el acoplamiento

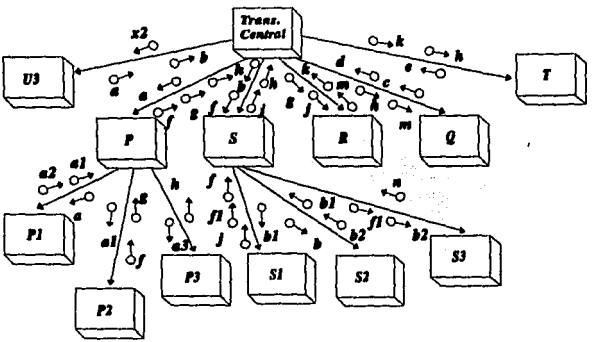


Fig. V.32 Diagrama de Estructura del Módulo de Transformación Central

necesario fragmentar más a los módulos, reagruparlos o moverlos de lugar en el diagrama de estructura con el objeto de maximizar su cohesión interna y minimizar el acoplamiento entre ellos.

Las ventajas principales del análisis de transformación son las siguientes:

- El diagrama de estructura se deriva de un procedimiento totalmente mecanizado.
- Conserva un fuerte parecido con el modelo representado por el diagrama de flujo de datos.
- Trata con grados variables de abstracción de datos, los más abstractos a nivel superior y los más físicos a nivel inferior.

V.3.2. Análisis de transacción.

No siempre es posible tomar como estrategia el análisis de transformación para obtener el diagrama de estructura del sistema. Una estrategia alterna que se puede utilizar es la del análisis de transacción, la cual está enfocada a diseñar el diagrama de estructura para un sistema que procese transacciones.

V.3.2.1. Principales usos.

La estrategia de análisis de transacción tiene dos usos principales:

- Para dividir un diagrama de flujo de datos grande y complicado en otros más pequeños y sencillos, uno para cada transacción que procese el sistema. Cada uno de los resultantes se convertirá en un diagrama de estructura de cada una de las transacciones.

- La estrategia se puede usar para combinar los diagramas de estructura individuales con cada tipo de transacción, en un diagrama de estructura general y muy flexible a los cambios de usuarios.

Antes de explicar el principio de este tipo de análisis es necesario definir lo que es una transacción.

Una transacción es un dato o colección de datos que pueden caer en una variedad de tipos. Cada tipo de transacción se procesa con un conjunto de actividades dentro del sistema.

Algunos ejemplos en el sistema de control de biblioteca para transacción sería dar de alta a un nuevo lector, darlo de baja, cambiarle sus datos, etc.

V.3.2.2. Principio del análisis de transacción.

El análisis de transacción se basa en el principio del uso de etiquetas para separar e identificar los diferentes tipos de transacciones en lugar de tomar en cuenta requerimientos comunes de proceso de cada tipo.

Por cada transacción que haya habrá un módulo que la procese. Además es importante mencionar al módulo centro de transacción, el cual determina el tipo de transacción que entra al sistema, inspecciona su etiqueta y la envía al módulo correspondiente.

En la figura V.33 se muestra un ejemplo de los módulos antes mencionados. El símbolo en forma de diamante indica a que módulo subordinado se va a llamar dependiendo de una prueba previa para identificar el tipo de transacción a procesar.

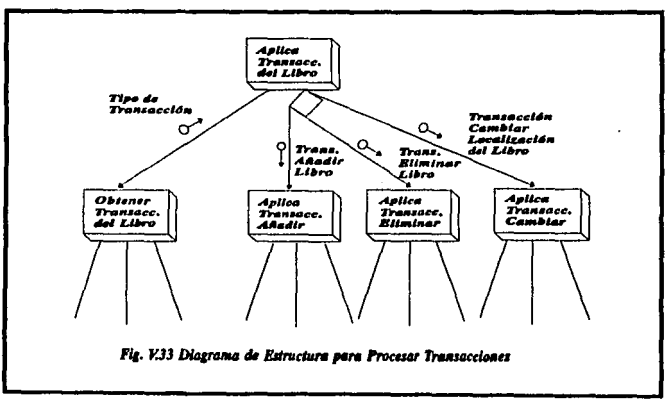


Fig. V.33 Diagrama de Estructura para Procesar Transacciones

V.3.2.3. Ventajas del análisis de transacción.

Las ventajas de esta estrategia para diseño estriban en el hecho de que un proceso de un tipo de transacción está totalmente separado de los demás, de esta forma, cualquier cambio en alguno de ellos no afectará a los otros.

La desventaja aparente es la duplicidad de código en muchos módulos diferentes, ya que algunos tipos de transacción requieren procesos similares. Sin embargo, una buena medida para eliminar tal situación, es la factorización de los módulos.

La factorización de módulos de este tipo consiste en extraerles las funciones comunes y dejarlas en otros, a un nivel inferior, así cada uno de estos será invocado según convenga.

El análisis de transacción conduce a sistemas bastante más generales y flexibles que el enfoque tradicional de procesar transacciones debido a que evita depender de similitudes que pueden cambiar.

V.4. DIAGRAMAS HIPO.

El método corriente o más común para el diseño de un sistema es utilizar el método de diseño "de arriba a abajo" Hipo-Hierarchy Plus Input Process-Output (Jerarquía más Entrada-Proceso-Salida). El nivel más alto en la jerarquía describe la función global que el sistema debe de realizar.

Luego se desarrollan niveles jerárquicos adicionales para describir el sistema con mayor detalle cada vez.

Los diagramas HIPO están proyectados de manera que suministren documentación en todos los niveles, para la administración, para el personal de sistemas, para los programadores, y aquellos implicados en el mantenimiento.

Los diagramas HIPO se componen de tres niveles:

V.4.1. Tabla Visual de Contenidos.

Suministra, de la manera más concisa posible las funciones del sistema. Es el Indice de la descripción del sistema.

La fig. V.34 es una parte de la Tabla de contenido de un sistema de información de un hospital. El cuadro superior indica la función global del sistema, esto es suministra la información acerca del paciente.

La siguiente linea de cajas o cuadros indica las subfunciones principales del sistema, conformar los registros, establecer el archivo, actualizar el archivo, suministrar o proporcionar respaldo, acumular un archivo en desuso, investigar el archivo y producir reportes

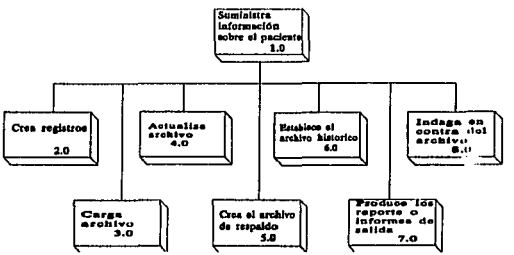


Fig. V.34 Tabla de contenidos del sistema de un hospital.

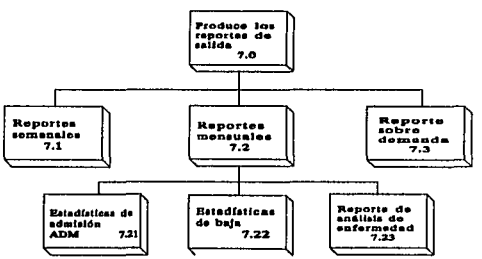


Fig. V.35 Tabla expandida del contenido, del sistema de un hospital.

de salida. Cada cuadro contiene un número de referencia que se refiere a diagramas detallados sobre esa función. Otro nivel de cuadro puede ser necesario para explicar con más detalle las seis funciones principales. En la fig. V.35 se ha extendido la función "produce reportes de salida" (7.0) para indicar que ciertos reportes son semanales, otros mensuales y algunos reportes son "sobre pedido". Una expansión posterior mostrará la función de cada reporte producido. La tabla de contenido es sólo un resumen del diseño global, que cambiará continuamente conforme se corrija el diseño y se mejore. En nuestro ejemplo, quizás el reporte de Análisis de enfermedad llegue a ser un reporte "sobre pedido".

V.4.2. Diagrama Global o General.

Muestran en general la entrada, la salida, las actividades de procesamiento que se realizaran en cada función principal. Usando flechas se muestra la relación de éstos para cada función. Algunas veces una flecha no muestra realmente cómo trabajará un paso en particular, en cuyo caso se hacen notas para complementar el Diagrama Global. Estas notas están contenidas en una hoja llamada de Descripciones Extendidas.

Ahora se necesita un Diagrama Global del sistema que muestre con más detalle cada una de aquellas funciones principales. la fig. V.36 muestra el diagrama del cuadro 1.0. el Doctor observará que hay 3 encabezados principales, en este tipo de diagrama, Entrada, Proceso y Salida, que son las tres operaciones fundamentales de procesamiento de datos. Este primer diagrama es muy sencillo y describe en español los procesos a ser llevado a cabo sobre las diversas entradas y cuales serán las salidas subsecuentes. Cada proceso está contenido en un casillero con un número que concuerda con una función en la tabla de contenido. También se encuentra como la identificación del diagrama (Diagrama ID) en un diagrama de detalle que muestra además los detalles de la función. El cuadro 7.0 se extiende o expande en niveles posteriores en la tabla de contenido, de manera que se lo

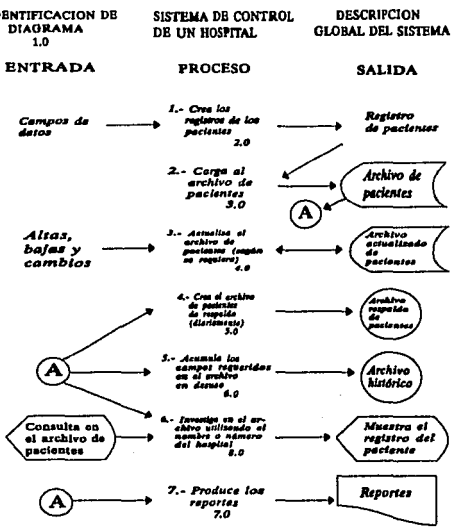


Fig. V.36 Diagrama Global del Sistema de un hospital.

expandirá un poco más y se examinará el diagrama global para la subfunción contenida en el casillero 7.0 (Fig. V.37). Un paso más nos dirá acerca de los reportes reales que se producirán (Fig. V.38). El siguiente nivel definirá completamente esos reportes.

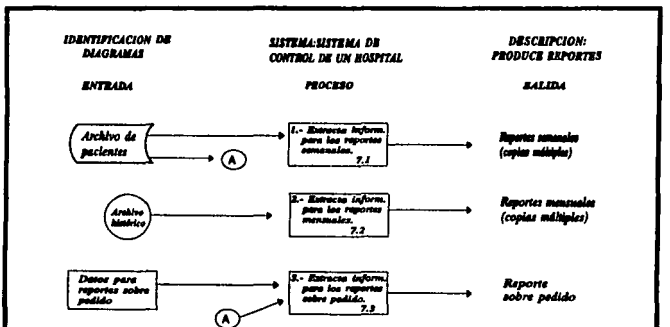


Fig. V.37 Diagrama General de sistema, con la función de producción de reportes expandidos.

V.4.3. Diagrama de Detalle.

Examínese el cuadro 7.23 (Fig. V.39). Este es un diagrama de detalle, se requerirán otros diagramas de detalle para algunas subfunciones.

En necesario que se muestre el suficiente detalle como para que permita al programador hacer una distribución de reporte. La distribución de reporte es parte de la documentación complementaria que casi siempre se necesita. Otros ejemplos de tal documentación serían la distribución de una pantalla. Puesto que se ha definido a un reporte de salida en algún

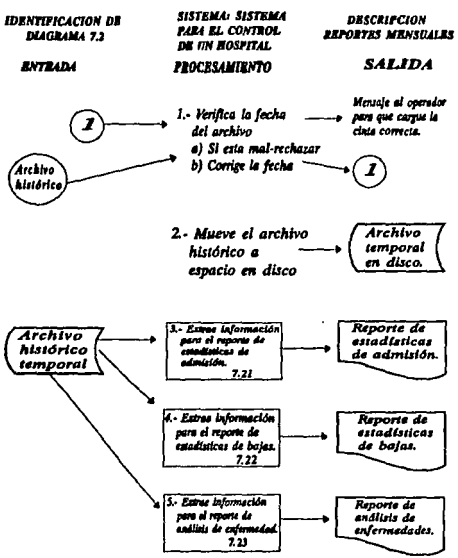


Fig. V.38 Diagrama global que muestra los reportes a ser producidos.

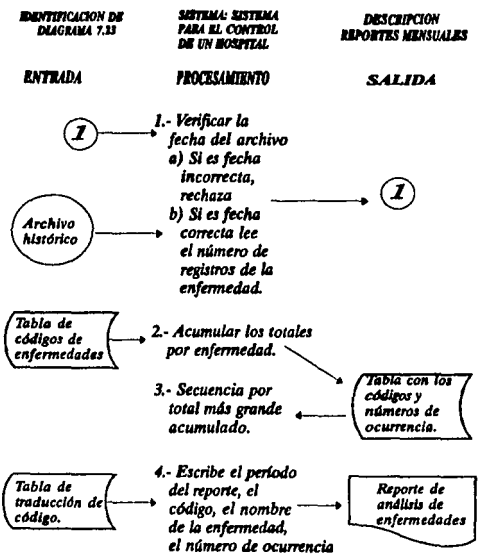


Fig. V.39 Diagrama de detalle, reporte de análisis de enfermedad-mensual.

detalle, el problema de diseñar la entrada al sistema (casillero 2.0) se hace mucho más fácil. Una vez que se han diseñado todos los reportes de salida, entonces se habrá identificado el contenido del archivo del paciente. La expansión del casillero 2.0 probablemente tendría lugar al mismo tiempo que la expansión del casillero 8.0, y también los casilleros 6.0 y 7.0. Conforme se establezcan más detalles se detectarán imperfecciones en el diseño y se les corregirá antes de que se desperdicie mucho tiempo. Por ejemplo, para producir todos los reportes que se solicitaron, al tiempo de estar suministrando también una función de investigación, puede no ser práctico con el equipo instalado y habrá más restricciones de dinero a la expansión. Los diagramas HIPO, junto con cualquier documentación complementaria requerida, representará la documentación completa del sistema real. En alguna fecha futura se puede decidir que ciertos reportes que ahora se producen mensualmente sean producidos sobre pedido. Por ejemplo, suponga que se ha hecho una solicitud para producir el reporte de Análisis de enfermedad sobre pedido; entonces el cuadro etiquetado 7.23 sería eliminado de la expansión del cuadro 7.2 y aparcería en la expansión del casillero 7.3 (Fig. V.35). El diagrama que representa la expansión de la casilla 7.23 se volvería 7.34, 7.35, o cualquiera que fuera el siguiente número disponible, y sería alterado de tal manera que se reflejara los cambios en el diseño del reporte. La figura V.40 muestra como a qué se parecerían los detalles del reporte si fuera un reporte sobre pedido. Se observará que ahora la entrada consiste de una serie de palabras que indican cuál sería la composición del reporte. El archivo en desuso habría de estar disponible todo el tiempo si el reporte se fuera a estar disponible "sobre pedido".

Quizá la ventaja más grande que se gana de este método de describir el diseño de un sistema es que la función principal (o funciones) del sistema es la primera fase decidida y la primera que se documenta. Cada vez que tiene lugar una revisión, es requisito principal del sistema que esté documentado y que esa documentación esté disponible. Además, los diagramas globales son comprensibles para el usuario. Un diagrama de flujo de sistema tradicional tiende a ser demasiado detallado y orientado hacia procesamiento de datos como para que un usuario que no sea de procesamiento de datos lo comprenda. No sólo

debe ser el usuario capaz de seguir el progreso, por lo tanto, ofrecer sugerencias conforme se desarrolla el sistema, sino que además se le suministrará al programador el suficiente detalle para que comience a programar, y sabiendo que los programas se acomodarán al sistema final.

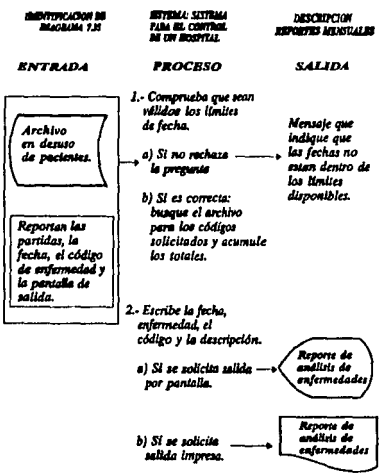


Fig. V.40 Diagrama de reporte de análisis de enfermedad "sobre pedido".

V.5. DIAGRAMAS DE WARNIER/ORR.

Los diagramas de Warnier/Orr (también conocidos como construcción lógica de programas/construcción lógica de sistemas) fueron desarrollados inicialmente en Francia por Jean-Dominique Warnier y en los Estados Unidos por Kenneth Orr. Este método ayuda al diseño de las estructuras de un programa identificando la salida y los resultados del procesamiento y después trabaja retrocediendo para determinar los pasos y combinaciones de entrada que se necesitan para producirlos. Los métodos gráficos simples utilizados en los diagramas de Warnier/Orr hacen evidentes los niveles del sistema y el movimiento de datos experimentado.

V.5.1 Notación.

Los diagramas de Warnier/Orr muestran los procesos y secuencias en los cuales se desempeñan. Cada proceso se define en forma jerárquica, es decir consiste en conjuntos de subprocesos que lo definen. En cada nivel el proceso se muestra en llaves que agrupan sus componentes (Fig. V.41). Dado que un proceso puede tener muchos subprocesos diferentes, un diagrama de Warnier/Orr es un conjunto de llaves que muestran cada nivel del sistema.

Los factores críticos en la definición del sistema y en el desarrollo son la iteración o repetición, y la intercalación. Los diagramas de Warnier/Orr lo muestran muy bien. Por ejemplo, al desarrollar un sistema de facturación existen procesos que ocurren para cada transacción cada día y cada mes. Como se muestran en la figura V.42, la repetición de cada caso no sólo es fácil de identificar sino también rápidamente se coloca en secuencia lógica.

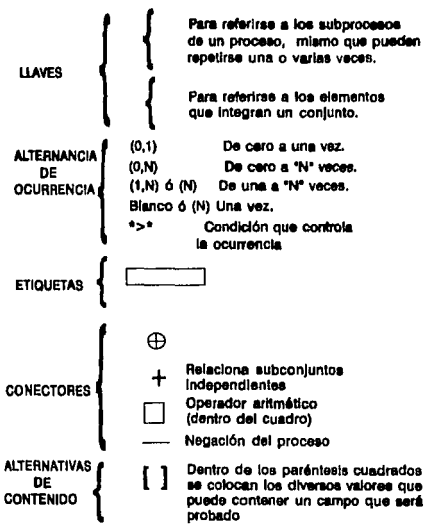
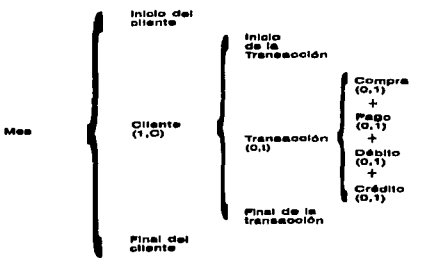


Fig. V.41 Símbolos Empleados en los Diagramas de Warnier/Orr.



(1,C) es el número de clients; por ejemplo, C es el número de clientes.
(1,t) es el número de transacciones; es decir, t es el número de transacciones.

Fig.42 Procesos Incluidos en el Procesamiento Mensual de Facturas.

V.5.2. Como Utilizar los Diagramas Warnier/Orr.

La capacidad de mostrar la relación entre los procesos y pasos en un proceso no es única en los diagramas de Warnier/Orr, como tampoco lo es el uso de la iteración, intercalación o tratamiento de casos individuales. Tanto los diagramas de flujo estructurados como los métodos de español estructurado lo hacen de igual manera. Sin embargo, el enfoque utilizado para desarrollar las definiciones de sistemas con los diagramas de Warnier/Orr es diferente y cumple muy bien con aquellos utilizados en el diseño de sistemas lógico.

Para desarrollar un diagrama de Warnier/Orr el analista trabaja retrocediendo para comenzar con la salida del sistema; es decir, este método utiliza un análisis orientado a la salida. Al representarlo sobre el papel, el desarrollo se mueve de izquierda a derecha.

Primero se define la salida deseada o los resultados del procesamiento. En el siguiente nivel, que se muestra con la inclusión de una llave, se definen los pasos necesarios a fin de producir la salida. Cada paso, a su vez, se define más adelante. Llaves adicionales agrupan los procesos requeridos para producir el resultado en el siguiente nivel.

Un diagrama completo de Warnier/Orr incluye tanto agrupamientos de procesos como requerimiento de datos. Como se indica en la figura V.43, los elementos de datos se listan para cada proceso o componente de proceso. Estos elementos de datos son los que se necesitan para determinar qué alternativa o caso debe ser manejado por el sistema y para llevar a cabo el proceso. el análisis debe determinar en dónde se origina cada elemento de datos, cómo se utiliza y cómo se combinan los elementos individuales. Cuando se completa la definición se documenta una estructura de datos para cada proceso, lo que, a su vez, es utilizado por lo programadores, quiénes trabajan a partir de los diagramas para codificar el software.

Los diagramas de Warnier/Orr ofrecen distintas ventajas a los expertos de sistemas. son simples en apariencia y fáciles de entender; además, son herramientas poderosas de diseño. Tienen la ventaja de mostrar agrupamientos de procesos y los datos y los datos que deben pasar de un nivel a otro; asímismo, la secuencia de trabajar retrocediendo asegura que el sistema estará orientado hacia los resultados, lo que es también un proceso natural. Con los diagramas de flujo estructurados, por ejemplo, a menudo es necesario determinar los pasos internos antes de que las iteraciones y aspectos de modularidad se puedan digerir.

Este método es útil tanto para la definición de datos como de proceso. Se puede utilizar cada uno en forma independiente o se pueden combinar ambos en un mismo diagrama, como se realizó en el ejemplo anterior.

Una característica poderosa de los diagramas de Warnier/Orr es la forma en la que se muestran los ciclos. Los pasos requeridos para moverse mes a mes al procesar facturas, por ejemplo, se señalan en el mismo diagrama. Los pasos de procesamiento y de

preparación de informes se señalan con claridad, de manera que la persona que utilice el diagrama pueda visualizar rápidamente cuando se están ejecutando los pasos y qué aspectos están preparados para terminar un ciclo mensual y prepararse para el siguiente. este aspecto del procesamiento de sistemas es crítico en un sistema de calidad.

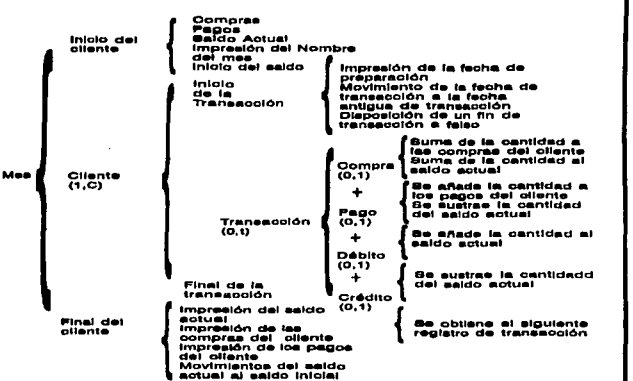


Fig. V.43 Datos Utilizados en el Procesamiento Mensual de Facturas.

CAPITULO VI

CODIFICACION Y LOS LENGUAJES DE PROGRAMACION

INTRODUCCION

Todos los pasos de la ingeniería del software, presentados hasta ahora van dirigidos hacia un objetivo final: traducir las representaciones del software a una forma que pueda ser "comprendida" por la computadora. Se ha llegado a la etapa de la codificación, un proceso que transforma el diseño en un lenguaje de programación, es decir, se implementa como un programa en un lenguaje específico. En esta etapa de desarrollo pueden intervenir los mismos programadores de etapas anteriores o nuevos, por esta causa, es importante que el diseño del software sea cuidadosamente documentado, basado en diagramas de estructuras y pseudocódigo.

Este capítulo presentará todos aquellos aspectos relacionados con los lenguajes de programación y con la codificación, más no pretende enseñar como programar.

VI.1. PROGRAMACION SISTEMATICA

Tras conocer las características que debe tener un programa, es necesario seguir una metodología encaminada al cumplimiento de esas características. La metodología de la programación es la técnica que permite que la programación sea lo más eficaz posible en cuanto al desarrollo y mantenimiento.

La realización de un programa sin seguir un método de programación riguroso aunque funcione, a la larga no será más que un conjunto más o menos grande de instrucciones. La consecuencia inmediata se podría recoger en la siguiente relación de problemas y defectos que suelen tener los programas escritos sin un método determinado:

- Los programas suelen ser excesivamente rígidos, con los consecuentes problemas para adaptarlos a las, cada día, más cambiantes configuraciones.
- Los programadores gastan la mayoría del tiempo corrigiendo sus errores.
- Los programadores generalmente rehúsan el uso de programas y módulos ya escritos y en funcionamiento, prefiriendo escribir los suyos.
- Un proyecto de varios programadores tienen normalmente varios conjuntos diferentes de objetivos.
- Cada programador tiene sus propios programas convirtiéndose esta relación en algo inseparable.
- Las modificaciones en las aplicaciones y programas son muy difíciles de hacer; implican mucho tiempo y elevado costo de mantenimiento.
- Deficiencias en la documentación.

Los problemas anteriores y otros no citados, no permiten la evolución y mantenimiento de los programas. Por consiguiente, es de suma importancia prever las futuras modificaciones a los programas. Estas previsiones se pueden resumir en:

- Aumento al volumen de datos y estructuras.
- Cambios en la organización de la información.
- Cambios debidos preferentemente a la modernización de los documentos y sus formatos.
- Sustitución, ampliación o reducción en el sistema de proceso de datos.

Así pues, se deben crear programas claros, inteligibles y breves con el objetivo de que puedan ser leídos, entendidos y fácilmente modificados. A la hora de diseñar un programa, éste debe reunir las siguientes características fundamentales:

- *Correcto/fiel:* producir resultados requeridos;
- *Legible*: debe ser entendido por cualquier programador;
- *Modificable:* el diseño nunca es definitivo por lo tanto su estructura debe permitir modificaciones;
- *Depurable:* debe ser fácil la localización y corrección de errores.

A continuación veremos las técnicas de programación modular y programación estructurada, las cuales nos permiten, llevar a cabo una metodología para la realización de programas más óptimos.

VI.1.1. Programación modular

La programación modular es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa. En la programación modular, el programa se divide en módulos (partes independientes), cada uno de los cuales ejecuta una única actividad o tarea y se codifican independientemente de otros. Cada uno de estos, se analizan, codifican y se ponen a punto por separado.

La división de un problema en módulos independientes, requiere de otro que controle y relacione a todos los demás, éste será llamado módulo base o principal del problema.

Las ventajas de la programación modular se pueden resumir en los siguientes puntos:

- Un programa modular es más fácil de escribir y depurar.
- Es más fácil de mantener y modificar.
- Es fácil de controlar, dividiéndolo en módulos complejos y sencillos.
- Posibilita el uso repetitivo de las rutinas en él mismo o diferentes programas.

Los inconvenientes se pueden resumir en:

- No se dispone de algoritmos formales de modularidad, por lo que a veces los programadores no tienen ideas claras de los módulos.
- La programación modular requiere más memoria y tiempo de ejecución.

Los objetivos de la programación modular se podrían resumir en los siguientes:

- Disminuir la complejidad.
- Aumentar la claridad y fiabilidad.
- Disminuir el costo.
- Aumentar el control del proyecto.
- Facilitar la ampliación del programa mediante nuevos módulos.
- Facilitar las modificaciones y correcciones.

VI.1.1.1. Concepto de módulo.

Un módulo está constituido por una o varias instrucciones físicamente continuas y lógicamente encadenadas, las cuales se pueden referenciar mediante un nombre y pueden ser llamadas de diferentes puntos de un programa. Un módulo puede ser:

- Un programa.
- Una función.
- Una subrutina.

Características de un módulo: Los módulos deben tener la máxima cohesión y el mínimo acoplamiento. Es decir, deben tener la máxima independencia entre ellos. La salida de un módulo debe estar en función de la entrada, pero no de ningún estado interno.

Requisitos de la programación modular: Con independencia de las técnicas, los requisitos que debe cumplir la programación modular son:

- Establecimiento de un organigrama modular.
- Descripción del módulo principal.
- Descripción de los módulos básicos o secundarios.
- Normas de programación.

El organigrama modular se realiza mediante bloques, en el que cada bloque corresponde a un módulo y muestra gráficamente la comunicación entre el módulo principal y los secundarios. El módulo principal debe ser claro y conciso, reflejando los puntos fundamentales del programa. Los módulos básicos deben resolver partes bien definidas del problema, ya que sólo pueden tener un punto de entrada y uno de salida. Las normas de programación dependerán del análisis de cada problema y de las normas generales o particulares que haya recibido el programador.

Clasificación de los módulos: Según las funciones que puede desarrollar cada módulo, éstos se clasifican en:

- Módulos tipo raíz, director o principal.
- Módulos tipo subraíz.
- Módulos de entrada (captura de datos).

- Módulos de variación de entradas.
- Módulos de proceso.
- Módulos de creación y formatos de salidas.

VI.1.1.2. Técnicas de la programación modular

Las fases de la resolución de un problema con programación modular son las siguientes:

- Estudio de las especificaciones del problema.
- Confección del ordinograma o tabla de decisión de cada módulo.
- Codificación de cada módulo en el lenguaje adecuado.
- Pruebas parciales de cada uno de ellos.
- Prueba final de los módulos enlazados.

La programación modular se basa en el diseño descendente (top-down) que permite comprobar el funcionamiento de cada módulo mediante otros ya comprobados (Fig. VI.1.).

VI.1.2. Programación Estructurada.

El término programación estructurada se refiere a un conjunto de técnicas, las cuales aumentan considerablemente la productividad del programa reduciendo el tiempo requerido para escribir, verificar, depurar y mantener los programas.

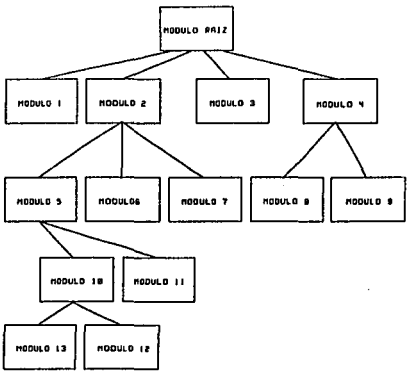


Fig. VI.1. Diseño descendente (top-down).

La programación estructurada es el conjunto de técnicas que incorporan:

- Estructuras básicas.
- Recursos abstractos.
- Diseño descendente "arriba-abajo" (top-down).

VI.1.2.1. Estructuras Básicas.

En mayo de 1966 Bohm y Jacopini demostraron que un programa propio puede ser escrito utilizando solamente tres tipos de estructuras de control:

- Secuencial;
- Alternativa;
- Repetitiva.

Estructura secuencial: Una estructura secuencial es aquella que ejecuta las acciones sucesivamente, unas a continuación de otras sin posibilidad de omitir ninguna y sin bifurcaciones. Todas estas estructuras tendrán una entrada y una salida. Un ejemplo de un programa que sólo contiene estructura secuencial se muestra en la Fig. VI.2.

Estructura alternativa: Es aquella estructura en la que únicamente se realiza una alternativa, dependiendo del valor de una determinada condición. Las estructuras alternativas, también llamadas condicionales, pueden ser de tres tipos:

- Simple;
- Doble;
- Múltiple.

Estructura Alternativa Simple: Es aquella en que la existencia o cumplimiento de la condición implica la ruptura de la secuencia y la ejecución de una determinada acción (Fig.VI.3.).

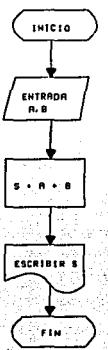


Fig. VI.2. Estructura secuencial

Estructura alternativa doble: Es aquella que permite la elección entre dos acciones o tratamientos en función de que se cumpla o no determinada condición (Fig. VI.4.).

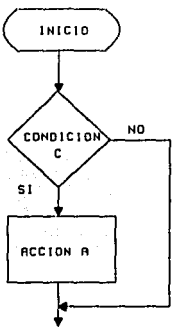


Fig. VI.3. Estructura alternativa simple.

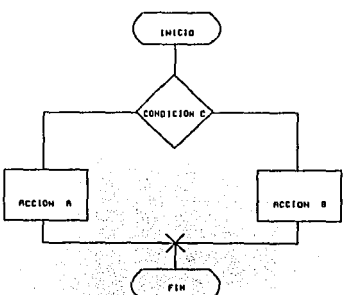


Fig. VI.4. Estructura alternativa doble.

Estructura alternativa múltiple: Las estructuras alternativas múltiples se adoptan cuando la condición puede tomar un número "n" de valores enteros distintos: 1, 2, 3, ..., n. Según se elija uno de estos valores en la condición, se realizará una de las "n" acciones, es decir cada vez sólo se ejecuta una acción (Fig. VI.5.).

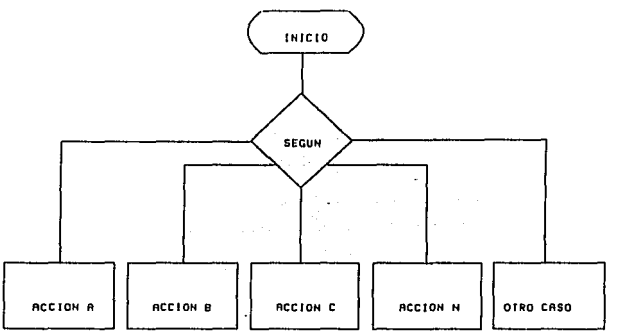


Fig. VI.5. Estructura alternativa múltiple.

Estructuras repetitivas: Las estructuras repetitivas son aquéllas en las que las acciones se ejecutan un número determinado de veces y dependen de un valor predefinido o el cumplimiento de una determinada condición. Estas estructuras repiten un número determinado de veces, por lo tanto se denominan *bucles* y se conoce como *iteración* a la acción (o acciones) que se repite dentro de un bucle.

Las tres estructuras más usuales, dependiendo de que la condición se encuentre al principio o al final del bucle, son:

- Estructura mientras ("while")
- Estructura repetir ("repeat")
- Estructura desde/para ("for")

Estructura mientras: La estructura repetitiva **mientras** (WHILE o DOWHILE: "hacer mientras"), determina la repetición de un grupo de instrucciones mientras la condición inicial se cumpla (Fig VI.6.).

Estructura repetir: Es la estructura en la que el número de repeticiones o iteraciones del grupo de instrucciones se ejecuta hasta que la condición deja de cumplirse. Esta condición se cumple al final (Fig. VI.7.).

Estructura desde/para: la estructura desde/para es aquella que se repite un número fijo de "n" veces (Fig. VI.8.).

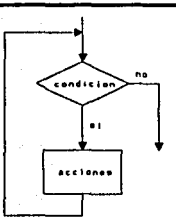


Fig. VI.6. Estructura mientras ("while").

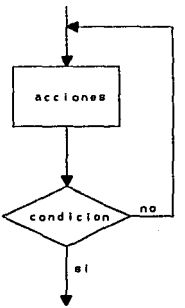


Fig. VI.7. Estructura repetir ('repeat').

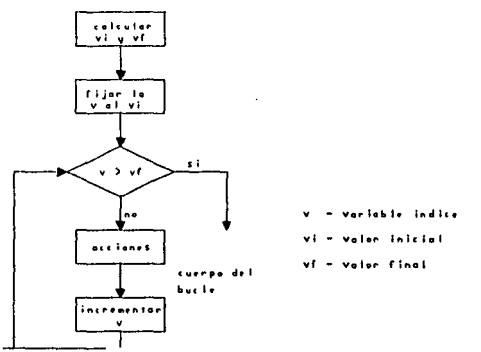


Fig. VI.8. Estructura desde/para ('for').

VI.1.2.2. Recursos abstractos

La programación estructurada se auxilia de los recursos abstractos en lugar de los concretos de que se dispone (un determinado lenguaje de programación o la máquina que lo va a resolver).

Descomponer un programa en términos de recursos abstractos, consiste en descomponer una determinada acción compleja en términos de un número de acciones más simples capaz de ser ejecutadas por la máquina tal y cómo han sido concebidas por el ordinograma o el algoritmo; si eso no fuera posible, será preciso descomponer las acciones en otras subacciones más elementales, continuándose el proceso hasta que cada subacción pueda ser codificada en el lenguaje elegido y por consiguiente ejecutado en la computadora.

VI.1.2.3. Metodología descendente "arriba-abajo"

La metodología o diseño descendente (top-down), también conocida como arriba-abajo, consiste en establecer una serie de niveles o pasos sucesivos de refinamiento (stepwise), de menor a mayor complejidad (arriba-abajo) que den solución al problema. En esencia, consiste en efectuar una relación entre etapas de la estructuración, de forma que una etapa jerárquica y su inmediatamente inferior se relacionen mediante entradas y salidas de información. Un grupo estructurado tiene una representación en forma de árbol como se ilustró en la Fig. VI.1.

VI.2. LAS HERRAMIENTAS DE PROGRAMACIÓN

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Esto permitirá que el algoritmo pueda ser codificado en cualquier lenguaje. Los métodos mas usuales para representar un algoritmo son:

- **Diagramas de flujo.**
- **Diagramas de N - S (Nassi - Schneiderman).**
- **Pseudocódigo (Lenjuage de representación de algoritmos).**

VI.2.1. Diagrama de flujo

Un diagrama de flujo (flowchart) es una de las técnicas de representación de algoritmos más antigua y a la vez más utilizada, aunque su empleo ha disminuido desde la aparición de los lenguajes de programación estructurada.

Un diagrama de flujo es un diagrama que utiliza los símbolos (cajas) estándar que se muestran en la Figura VI.9. y que tienen los pasos del algoritmo escritos en esas cajas unidas por flechas, denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar.

La Fig. VI.10. muestra un ejemplo de un diagrama de flujo, para calcular el factorial de un número natural.

En los diagramas de flujo se pueden escribir más de un paso del algoritmo en una sola caja rectangular. El uso de flechas significa que la caja no necesita ser escrita debajo de su predecesora. Sin embargo, abusar demasiado de esta flexibilidad conduce a diagramas de flujo complicados e ininteligibles.

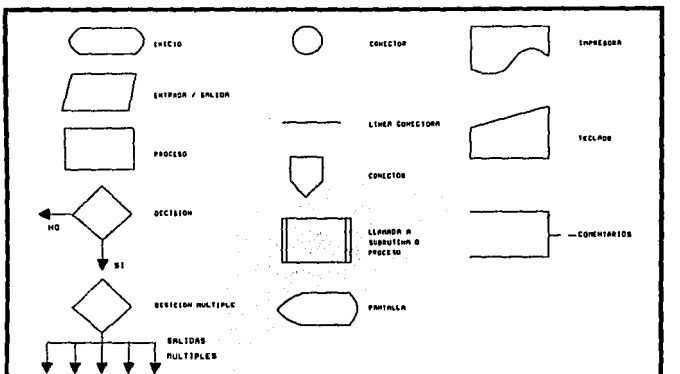


Fig. VI.9. Simbolos para diagramas de flujo.

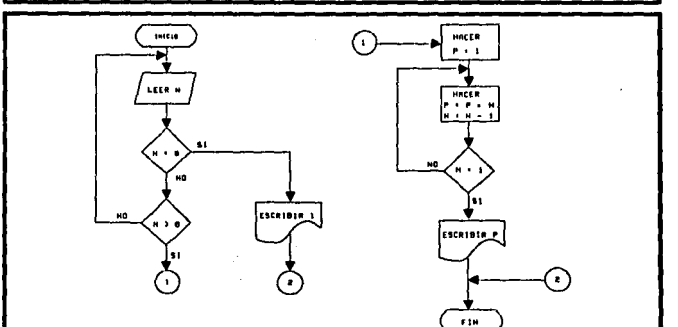


Fig. VI.10. Ejemplo de un diagrama de flujo.

VI.2.2. Diagramas de Nassi-Schneiderman

Un enfoque más estructurado, pero tal vez menos visual para el diseño de programas es el diagrama de Nassi-Schneiderman (N-S). La principal ventaja de un diagrama N-S es que adopta la filosofía de la programación estructurada. Por otro lado, utiliza un número limitado de símbolos, de tal forma que el diagrama ocupa menos espacio y puede leerse con cierta facilidad por la gente poco familiarizada con símbolos ajenos a los de los diagramas de flujo.

La Fig. VI.11. muestra los tres símbolos básicos que se utilizan en un diagrama N-S. El primero es un cuadro, que sirve para representar cualquier proceso en el programa. El segundo símbolo es una columna dividida por un triángulo incorporado, que representa una decisión. El tercero es un cuadro dentro de otro cuadro, que se utiliza para indicar que se lleva a cabo una repetición.

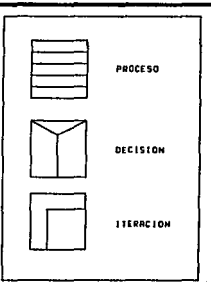


Fig. VI.11. Símbolos básicos de diagramas N - S.

En al Fig. VI.12. se ilustra un sistema de actualización de suscripciones de periódicos por medio de un diagrama N-S.

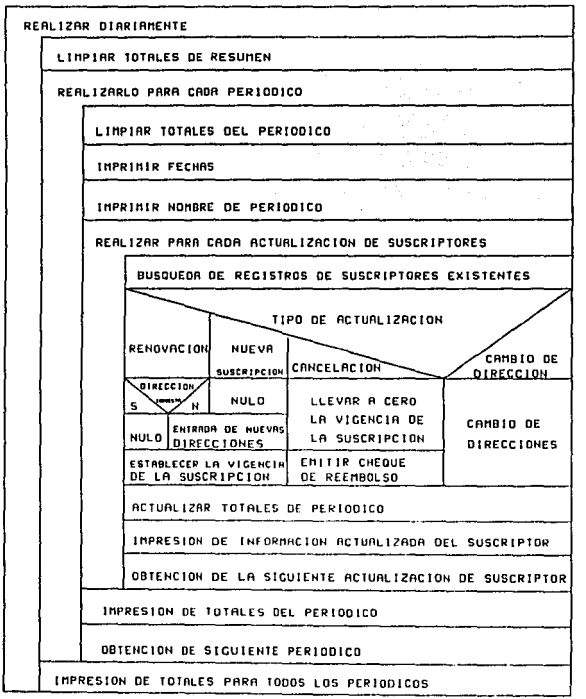


Fig. VI.12. Uso de un diagrama Nassi - Schneiderman.

Los diagramas N-S deben estar completos y ser muy claros, con el fin de que se entiendan. Si de manera regular se hacen cambios, los diagramas Nassi-Schneiderman no serán apropiados, ya que deberán dibujarse nuevamente; por lo tanto su modificación no es sencilla.

VI.2.3. Pseudocódigo

El pseudocódigo es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final relativamente fácil. El lenguaje APL se utiliza a veces como un lenguaje de especificación de algoritmos.

El pseudocódigo nació como un lenguaje similar al inglés y era un medio de representar básicamente las estructuras de control de programación estructurada. La ventaja del pseudocódigo es que su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Otra gran ventaja es que puede ser traducido fácilmente a lenguajes como Pascal, COBOL, C, FORTRAN 77 o BASIC estructurado.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés, tales como star, end, if-then-else, while-wend, repeat-until... etc. La escritura del pseudocódigo exige normalmente la identación (sangría en el margen izquierdo) de diferentes líneas. La Fig. VI.13. ilustra un sencillo ejemplo que representa la resolución de un programa que deduce el salario neto de un trabajador a partir de la lectura del nombre, horas trabajadas, precio de la hora, y sabiendo que los impuestos aplicados son el 10% sobre el salario bruto.

```
star
   {cálculo de impuestos y salarios}
   read nombre, horas, precio_hora
   salario_bruto = horas * precio_hora
   tasa = 0.10 * salario_bruto
   salario_neto = salario_bruto - tasa
   write nombre, salario_bruto, tasa, salario_neto
end
```

Fig. VI.13. Ejemplo para representación del pseudocódigo

VI.3. CLASES Y CARACTERÍSTICAS DE LOS LENGUAJES DE PROGRAMACIÓN

VI.3.1. Características de los lenguajes de programación

Los lenguajes de programación son un vehículo de comunicación entre los humanos y las computadoras, es por ello que las características psicológicas del lenguaje afectan directamente a la calidad de la comunicación. El proceso de codificación es una etapa más en la metodología de la ingeniería de software, las características de ingeniería de un

lenguaje tienen un impacto importante sobre el éxito en el desarrollo de un sistema de programación.

VI.3.1.1. Una visión psicológica

Los diseñadores de lenguajes de programación a menudo hacen que tengamos que ajustar nuestra aproximación a un problema de forma que tenga en cuenta las limitaciones impuestas por ese determinado lenguaje. Debido a que los factores humanos son de importancia crítica para el diseño de lenguajes de programación, las características psicológicas, tienen una fuerte influencia en el éxito del diseño, la traducción de código y la implementación.

Varias características psicológicas surgen como resultado del diseño de un lenguaje y aunque no se pueden medir cuantitativamente, se reconoce su manifestación en todos los lenguajes de programación. A continuación se presentan brevemente esas características.

La UNIFORMIDAD indica el grado en que un lenguaje usa una notación consistente, aplica restricciones aparentemente arbitrarias o incluye excepciones a reglas sintácticas semánticas. Por ejemplo, FORTRAN usa el paréntesis como delimitador para índices de arreglos, como modificador de la precedencia aritmética y como delimitador para las listas de argumentos de subprogramas. Esa notación multiuso ha llevado a más de un error.

La AMBIGÜEDAD de un lenguaje de programación es percibida por el programador, es decir un compilador siempre interpreta una sentencia de una única forma, pero el lector humano puede interpretar la sentencia de formas diferentes. Por ejemplo, la ambigüedad psicológica aparece cuando la precedencia aritmética no es obvia:

```
X = X1/X2 *X3
```

Un lector del código fuente puede interpretar lo anterior como X = (X1/X2)*X3, mientras que otro puede "ver" X = X1/(X2*X3). Una falta de uniformidad y la ocurrencia de ambigüedad psicológica normalmente se dan juntas.

Lo COMPACTO que sea un lenguaje de programación es un indicativo de la cantidad de información orientada al código que se debe retener en la memoria humana. Entre los atributos del lenguaje que miden lo compacto que es, se encuentran:

- El grado en que un lenguaje soporte las construcciones estructuradas.
- Los tipos de palabras clave y de abreviaturas que se pueden usar.
- La variedad de tipos de datos y las características implícitas.
- El número de operadores aritméticos y lógicos.
- El número de funciones incorporadas.

APL es un lenguaje de programación excepcionalmente compacto, sus concisos y potentes operadores permiten implantar importantes procedimientos aritméticos y lógicos con relativamente poco código. Desgraciadamente, al ser tan compacto, APL también es un lenguaje difícil de leer y de comprender, pudiendo llevar a una pobre uniformidad.

Las características de la memoria humana tienen un fuerte impacto en la forma en que se usa un lenguaje, la memoria y el reconocimiento humano se pueden dividir en dominio sinestético y secuencial. La memoria sinestética nos permite recordar y reconocer las cosas como un todo, es decir podemos reconocer una cara humana instantáneamente; sin necesidad de evaluar concisamente cada una de sus partes antes de reconocerla. La memoria secuencial proporciona una forma de reconocer el siguiente elemento de una secuencia, proporcionando un elemento precedente (por ejemplo la siguiente línea de

una canción, dadas las líneas precedentes). Cada una de estas características de la memoria afecta a las características de los lenguajes de programación denominadas localización y linealidad.

La localización es una característica sinestética de un lenguaje de programación, la linealidad es una característica psicológica que se asocia con el concepto de mantenimiento de un dominio funcional. O sea, la percepción humana se facilita cuando se encuentra una secuencia lineal de operadores lógicos. Las grandes ramificaciones y de alguna forma los grandes bucles , violan la linealidad del procesamiento. De nuevo, la implementación directa de la construcciones estructuradas ayudan a la linealidad de un lenguaje de programación.

Nuestra capacidad para aprender un nuevo lenguaje de programación se ve afectada por la tradición, un ingeniero de programación con experiencia en FORTRAN o PL/1 no tendrá mucha dificultad para aprender C o Pascal. Sin embargo, si se requiriera al mismo individuo a aprender APL o Lisp, se rompería la tradición y la curva del tiempo de aprendizaje sería más pronunciada.

La tradición también afecta al grado de innovación durante el diseño de un lenguaje. Aunque frecuentemente se proponen nuevos lenguajes, las formas de estos, evolucionan muy despacio. Por ejemplo, Pascal es un pariente cercano al ALGOL, Ada un lenguaje que ha crecido desde la tradición ALGOL-Pascal, se extiende más allá de esos lenguajes con una gran variedad de estructuras y tipos innovadores. Las características psicológicas de los lenguajes, tienen una importancia asociada a nuestra capacidad de aprenderlos, de aplicarlos y de mantenerlos.

VI.3.1.2. Una visión de ingeniería

Una visión de ingeniería del software sobre las características de los lenguajes de programación se centra en las necesidades que puede tener un proyecto específico. Aunque se podrían derivar esotéricos requerimientos para el código fuente, podemos establecer un conjunto general de características de ingeniería:

- Facilidad de traducción del diseño al código.
- Eficiencia del compilador.
- Portabilidad del código fuente.
- Disponibilidad de herramientas de desarrollo
- Facilidad de mantenimiento.

El grado de facilidad de la traducción del diseño al código proporciona una indicación de cómo se aproxima un lenguaje de programación a la representación del diseño.

Aunque los rápidos avances en velocidad del procesador y densidad de memoria han comenzado a disminuir la necesidad de "código super-eficiente", muchas aplicaciones todavía requieren programas rápidos y "ajustados" (requerimiento de poca memoria). Las críticas actuales a los compiladores de lenguajes de alto orden van dirigidas a la incapacidad de producir código ejecutable rápido y ajustado. Si el rendimiento del software es un requerimiento crítico, los lenguajes con compiladores optimizados pueden ser más atractivos.

La portabilidad del código fuente es una característica de los lenguajes, que se pueden interpretar de tres formas:

- El código fuente puede ser transportado de un procesador a otro y de un compilador a otro sin ninguna o muy pocas modificaciones.

- El código fuente permanece inalterado cuando cambia su entorno de funcionamiento (por ejemplo, cuando se instala una nueva versión de sistema operativo).

- El código fuente puede ser integrado en diferentes paquetes de software sin que prácticamente se requieran modificaciones debidas a las características propias del lenguaje de programación.

De las tres interpretaciones de portabilidad, la primera es con mucho la más frecuente, la estandarización continúa siendo el principal esfuerzo para la mejora de la portabilidad de los lenguajes. Si la portabilidad es un requerimiento crítico, se debe restringir el código fuente al estándar ISO (Organización Internacional de Estándares) o ANSI (Instituto Nacional Americano de Estándares), aunque existan otras posibilidades.

La disponibilidad de herramientas de desarrollo puede acortar el tiempo requerido para la generación del código fuente y puede mejorar la calidad del código. Muchos lenguajes de programación pueden ser adquiridos con un conjunto de herramientas que incluyen:

- Compiladores con depuradores.

- Ayudas de formato para el código fuente.

- Facilidades de edición incorporadas.

- Herramientas para control del código fuente.

- Extensas bibliotecas de subprogramas para una gran variedad de áreas de aplicación.
- Compiladores cruzados para desarrollo de microprocesadores y otras.

La facilidad de mantenimiento del código fuente es críticamente importante. El mantenimiento no se puede llevar a cabo hasta que no se entienda la estructura del sistema. Anteriores elementos de configuración del software (por ejemplo, documentación de diseño) proporcionan un fundamento para la facilidad de comprensión, ya que el código fuente final debe ser leído y modificado de acuerdo a los cambios en el diseño. Además, las propias características de documentación de un lenguaje (como son, longitud de identificadores no limitada, etiquetado, definición de estructuras y tipos de datos) tienen una fuerte influencia sobre el mantenimiento.

VI.3.1.3. Elección de un lenguaje

La elección de un lenguaje de programación para un proyecto específico se debe tener en cuenta tanto las características de ingeniería como las psicológicas. Entre los criterios que se aplican durante la evaluación de los lenguajes disponibles están:

- Area de aplicación general.
- Complejidad algorítmica y computacional.
- Entorno en el que se ejecutará el software.
- Consideraciones de rendimiento.

- Complejidad de las estructuras de datos.
- Conocimiento de la plantilla de desarrollo de software.
- Disponibilidad de un buen compilador.

A menudo C es un lenguaje elegido para el desarrollo de software de sistemas, mientras que lenguajes como Ada, C y Modula-2 (junto con FORTRAN y emsamblador) se encuentran en aplicaciones de tiempo real. COBOL es el lenguaje para aplicaciones de negocios, aunque el uso creciente de lenguajes de cuarta generación puede algún día desplazarlo de su posición privilegiada.

En el área científica/ingeniería, FORTRAN permanece como el lenguaje predominante. El lenguaje predominante para usuarios de computadoras personales es el BASIC, aunque este lenguaje es raramente usado en el desarrollo de productos de software para computadoras personales -opciones más usuales son Pascal o C-. Las aplicaciones de Inteligencia Artificial usan lenguajes como Lisp o PROLOG, aunque igualmente se usan lenguajes mas convencionales.

No importando el paradigma empleado de la ingeniería del software, el lenguaje de programación impactará la planificación, análisis, diseño, codificación, prueba y mantenimiento de un proyecto, los lenguajes proporcionan los medios de traducción hombre-máquina; sin embargo la calidad del resultado final se encuentra más fuertemente unida a las actividades de la ingeniería del software que preceden y siguen a la codificación.

Durante el paso de planificación del proyecto, raramente se toman en consideración las características técnicas de un lenguaje, sin embargo, una vez que se han establecido los requerimientos del software, las características técnicas de los lenguajes candidatos se hacen más importantes. Si se requieren complejas estructuras de datos, habrá que buscar un lenguaje que soporte esas estructuras (como Pascal o C), si lo importante es un alto rendimiento y posibilidades de tiempo real se debe especificar uno que este diseñado para

aplicaciones en tiempo real (por ejemplo Ada) o para eficiencia en memoria-velocidad (por ejemplo FORTH). Si se especifican muchos informes de salida y una fuerte manipulación de archivos, se encontrarán adecuados lenguajes como COBOL o RPG.

VI.3.2. Clases de lenguajes

Existen cientos de lenguajes de programación que han sido aplicados en uno u otro momento a la hora de realizar un desarrollo de software. En este punto se describen cuatro generaciones de lenguajes de programación, discutiendo los lenguajes más representativos de cada generación. La Fig. VI.14. ilustra la evolución histórica de los lenguajes de programación.

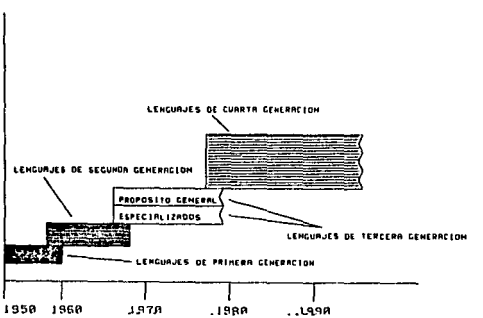


Fig. VI.14. Generaciones de los lenguajes de programación.

VI.3.2.1. Primera generación de lenguajes

La primera generación de lenguajes se remonta a los días en que se codificaba a nivel de máquina. El código máquina y su equivalente más humanamente legible, es el emsamblador, representa la primera generación de lenguajes.

Existen tantos lenguajes emsambladores como arquitectura de procesadores con sus correspondientes conjuntos de instrucciones desde un punto de vista de la ingeniería del software, estos lenguajes sólo se deben usar cuando un lenguaje de alto orden no satisfaga los requerimientos o no esté disponible.

VI.3.2.2. Segunda generación de lenguajes

La segunda generación de lenguajes fue desarrollada a finales de los 50's y principios de los 60's y ha servido como base de todos los lenguajes de programación modernos. La segunda generación está caracterizada por su amplio uso, la enorme cantidad de bibliotecas de software y la gran familiaridad y aceptación. Hay pocas discusiones sobre que FORTRAN, COBOL, ALGOL y BASIC son lenguajes de base debido a su madurez y aceptación.

FORTRAN a permanecido y sigue siendo el primer lenguaje de programación en el ambiente científico y de ingeniería. En muchos casos FORTRAN a sido forzado a ajustarse a áreas de aplicación para las que no fue diseñado, por lo que muchas de las críticas que ha recibido el lenguaje han sido injustas. Para las aplicaciones de cálculo numérico, FORTRAN sigue siendo el lenguaje elegido pero para aplicaciones de software de sistemas de tiempo real o de productos empotrados, otros lenguajes tienen ventajas sobre él.

COBOL, es el lenguaje aceptado como "estándar" para aplicaciones de procedimientos de datos comerciales. Fue diseñado y refinado como un lenguaje que hiciera que los programas y técnicas de programación fueran fácilmente compartidos y transferidos de unas máquinas a otras.

El área de aplicación para la que se diseño el COBOL es el área del procesamiento de datos, puesto que contiene poderosos elementos funcionales incorporados, tales como una característica de escritura uniforme, una función de búsqueda en tabla y una facilidad para la ordenación.

ALGOL es el predecesor de muchos lenguajes de tercera generación y ofrece un repertorio rico de construcciones procedimentales y de tipificación de datos. ALGOL es un lenguaje algebraico y lógico, se utiliza para presentar en forma precisa y estandarizada procedimientos numéricos a una computadora, permite la expresión concisa y eficiente de procesos aritméticos y lógicos, así como su control.

ALGOL soporta el concepto de estructuración por bloques, de asignación dinámica de memoria, de recursividad y otras características con gran influencia en los lenguajes modernos que le siguieron.

BASIC es el lenguaje más utilizado en computadoras personales, es un lenguaje originalmente diseñado para enseñar programación. Es relativamente compacto y no es difícil de aprender o de comprender. Existen cientos de versiones de BASIC, haciendo difícil discutir las ventajas y las deficiencias del lenguaje.

VI.3.2.3. Tercera generación de lenguajes

Los lenguajes de la tercera generación están caracterizados por sus potentes posibilidades procedimentales y de estructuración de datos. Los lenguajes de esta clase se pueden dividir en dos amplias categorías, lenguajes de alto orden de propósito general y lenguajes especializados.

VI.3.2.3.1. Lenguajes de alto orden de proposito general

El lenguaje de alto orden de propósito general más antiguo, ALGOL, ha servido como modelo para otros lenguajes de esta categoría. Sus descendientes, PL/1, Pascal, Modula-2, C y Ada, han sido adoptados como lenguajes con potencial para un gran espectro de aplicaciones.

PL/I, fue el primer lenguaje de amplio espectro, desarrollado con un amplio rango de posibilidades que le permiten ser usado en muchas áreas de aplicación. Se han desarrollado subconjuntos del lenguaje para enseñar programación (PL/C), para su uso en aplicaciones de microprocesadores (PL/M) y para programación de sistemas (PL/s).

Desde su introducción, Pascal es usado ampliamente en aplicaciones de ciencia e ingeniería y de programación de sistemas. Pascal es un descendiente directo del ALGOL y contiene muchas de sus propias características: estructuración en bloques, fuerte tipificación de datos, soporte directo de la recursividad y otras características suplementarias. Ha sido implementado en computadoras de todo tipo.

Modula-2 es un lenguaje evolucionado de pascal y una posible alternativa para el lenguaje de programación Ada. Modula-2 conecta las posibilidades de implementación directa del diseño, como el ocultamiento de información, la abstracción y la fuerte

tipificación de datos, con las estructuras de control que soportan la recursividad y la concurrencia.

El *lenguaje de programación C* fue originalmente desarrollado para la implementación de sistemas operativos. El sistema operativo Unix está implementado en C. El lenguaje C fue desarrollado para el ingeniero de programación sofisticado y contiene potentes posibilidades que le dan una considerable flexibilidad.

C soporta estructuras de datos sofisticados y tiene características de tipificación, hace uso intensivo de los punteros y tiene un rico conjunto de operadores para el cálculo y la manipulación de datos. Además permite al programador "acercarse a la máquina" al suministrar posibilidades similares al lenguaje emsamblador.

Ada es un lenguaje similar al Pascal en notación y estructura (aunque más potente y complejo), Ada soporta un rico conjunto de posibilidades que incluyen la multitarea, el manejo de interrupciones, la sincronización y la comunicación entre tareas, así como un conjunto de características únicas.

VI.3.2.3.2. Lenguajes especializados

Los lenguajes especializados están caracterizados por su usual formulación sintáctica que ha sido especialmente diseñada para una aplicación particular. En general estos lenguajes tienen una base de usuarios mucho menor que los lenguajes de propósito general. Entre los lenguajes que han encontrado aplicación en la ingeniería del software están Lisp, PROLOG, Smalltalk, APL y FORTH.

Lisp (List Processor) es un lenguaje diseñado originalmente para la manipulación de fórmulas simbólicas y el procesamiento de listas en problemas combinatorios. Usado casi exclusivamente por la comunidad de inteligencia artificial, el lenguaje es especialmente

adecuado para la prueba de teoremas, para búsquedas en árboles y otras actividades de resolución de problemas.

Lisp ha sido usado para el desarrollo de una cantidad de sistemas expertos y de "compiladores" de sistemas expertos. Lisp hace que sea relativamente fácil especificar hechos, reglas y su correspondiente inferencia (implementadas como funciones Lisp), lo que constituye la base de los sistemas basados en el conocimiento.

PROLOG (Programming in Login) diseñado principalmente para las aplicaciones de inteligencia artificial, el estilo de PROLOG se basa en la noción de definir objetos y relaciones de inferencia entre clases de objetos. Tanto Lisp como PROLOG son especialmente usados para los problemas que tratan objetos, por esta razón, alguna gente se refiere a éstos como lenguajes orientados a objetos.

Smalltalk, fue uno de los primeros lenguajes realmente orientados a objetos, introduce una estructura de datos y control radicalmente diferente de los lenguajes de programación convencionales. Smalltalk permite al programador definir objetos que combinan una estructura de datos y un conjunto de métodos. Cada objeto es una instancia de una clase de objetos, por lo tanto se pueden crear fácilmente nuevos objetos que hereden todas las características de su clase. Muchos creen que la aproximación orientada al objeto liderada por Smalltalk puede llevar a la creación de componentes de programa reusables que reducirían enormemente el tiempo de desarrollo para los grandes sistemas de software.

APL (A Programming Languaje) fue diseñado principalmente para los problemas que hacen un gran uso de tablas, vectores y matrices y para los que se desea una solución rápida, interactiva. El alfabeto del APL contiene un gran número de símbolos especiales, distintos del conjunto normal de caracteres ASCII o EBCDIC, añadidos para realizar una rápida especificación de poderosas funciones sobre arrays de números y cadenas de caracteres.

FORTH es un lenguaje diseñado para el desarrollo de software de microprocesadores, el lenguaje soporta la definición de funciones de usuario que se ejecutan de forma orientada a pila proporcionando una gran eficiencia en velocidad y utilización de memoria.

Desde el punto de vista de la ingeniería de programación, los lenguajes tienen tantas ventajas como desventajas, debido a que cada lenguaje se ha diseñado para una aplicación específica, se puede facilitar la traducción de los requerimientos del diseño a la implementación en código. Por otro lado, la mayoría de los lenguajes especializados no son portables y normalmente son menos fáciles de mantener que los lenguajes de propósito general.

VI.3.2.4. Lenguajes de cuarta generación

Los lenguajes de cuarta generación, al igual que los lenguajes de inteligencia artificial, contienen una sintaxis distinta para la representación del control y para la representación de las estructuras de datos. Sin embargo, un L4G representa estas estructuras en un mayor nivel de abstracción eliminando la necesidad de especificar detalles algorítmicos.

Los lenguajes de cuarta generación combinan características procedimentales y no procedimentales. O sea, el lenguaje permite al usuario especificar condiciones con sus correspondientes acciones (componente procedimental) mientras que pidiendo al mismo tiempo al usuario que indique el resultado deseado (componente no procedimental) para encontrar los detalles procedimentales aplicando su conocimiento del dominio específico.

VI.3.2.4.1. Lenguajes de petición

Hasta ahora, la gran mayoría de los L4G se han desarrollado para ser usados conjuntamente con aplicaciones de bases de datos. Tales lenguajes de petición permiten al usuario manipular de forma sofisticada la información contenida en bases de datos previamente creadas. Algunos lenguajes de petición tienen una sintaxis compleja que no es más sencilla que la de los lenguajes de tercera generación.

VI.3.2.4.2. Generadores de programas

Un generador de programas permite al usuario crear programas en un lenguaje de tercera generación usando notablemente menos sentencias. Estos lenguajes de programación de muy alto nivel hacen un fuerte uso de la abstracción de datos y de procedimientos.

Aunque los lenguajes de petición y los generadores de programas son los L4G más comunes, existen otras categorías. Los lenguajes de soporte a la toma de decisiones permiten que los no programadores lleven a cabo una gran variedad de análisis qué van desde los simples modelos de hojas de cálculo bidimensionales hasta los sofisticados sistemas de modelos estadísticos y de investigación operativa. Los lenguajes de prototipos se han desarrollado para asistir en la creación de prototipos facilitando la creación de interfases para el usuario y de diálogos, además de proporcionar medios para el modelado de datos. Los lenguajes de especificación formal se pueden considerar L4G cuando producen código máquina ejecutable.

VI.4. HERRAMIENTAS DE PUESTA A PUNTO DE PROGRAMAS

La fase de puesta a punto de un programa consiste en localizar, verificar y corregir los errores de programación para obtener un programa que funcione correctamente. La puesta a punto de un programa consta de las siguientes fases:

- Detección de errores;
- Depuración de errores;
- Localización y Eliminación;
- Prueba del programa.

El objetivo final de la puesta a punto de un programa será prevenir tantos errores como sea posible a la hora de ejecutar un programa, así como facilitar la detección y corrección de errores.

Los errores típicos durante la ejecución de un programa son:

- Errores de sintaxis
- Errores de lógica.

VI.4.1. Errores de sintaxis

Normalmente los errores de sintaxis se descubren con facilidad, ya que se originan en la fase de compilación del programa y se deben a causas propias de la sintaxis del lenguaje como escrituras incorrectas de instrucciones, omisión de signo, etc. En ocasiones puede suceder que un error de sintaxis, no sea detectado como tal, sino que se pone de manifiesto como otro tipo de error en la etapa de ejecución del programa. Por ejemplo, si en una instrucción del tipo M/N*P se omite el símbolo de división (/) y la instrucción se convierte en MN*P, la omisión de esté símbolo no se reconoce como un error de sintaxis en la fase de compilación o interpretación, ya que MN se considera como una variable y en la mayoría de los compiladores se producirá un error durante la ejecución, ya que la variable MN no ha sido declarada anteriormente.

VI.4.2. Errores de lógica

Los errores de lógica, suceden durante la ejecución de un programa, los cuales son más difíciles de detectar que los errores sintácticos.

Se pueden considerar dos categorías de errores: los que detienen la ejecución del programa y los que no la detienen, pero producen resultados erróneos. Los errores de lógica se pueden producir durante la fase de compilación o ejecución. Un ejemplo claro de error lógico puede ser:

```
A = M/N
```

Si N es cero, muchos de los errores lógicos sólo se manifiestan en el momento de la ejecución; por ello se denominan también errores de ejecución.

Prueba de programas.- Para saber si un programa funciona correctamente, será preciso realizar pruebas con conjuntos de datos de muestra, cuya solución sea conocida y correcta.

Validación de datos.- La validación de datos supone la verificación de que los datos de entrada son correctos y están dentro del rango válido.

CAPITULO VII

DOCUMENTACION

INTRODUCCIÓN

Hasta hace algunos años la documentación de sistemas informáticos, incluidos los manuales del usuario, estaba destinada a científicos, ingenieros y técnicos especializados. Por tal razón, la mayoría de las veces no se detallaban completamente todas las especificaciones, porque se suponía que se sobreentendían.

En nuestros días, la documentación de sistemas informáticos está dirigida a profesionales de negocios, oficinistas o personal administrativo, que por lo regular no estaban acostumbrados al tipo de información, que se hacia antes.

La importancia de la documentación bien podría ser comparada con la importancia de la existencia de una póliza de seguros: mientras todo va bien no existe la preocupación de confirmar si nuestra póliza de seguros está o no vigente, pero cuando se presenta una catástrofe o bien un daño nos es ocasionado, el resarcimiento económico estará dependiendo de la vigencia de esa póliza.

La documentación adecuada y completa, de una aplicación que se desea implantar, mantener y actualizar en forma satisfactoria, es esencial en cualquier sistema de programación, sin embargo, frecuentemente es la parte a la cual se dedica el menor tiempo y se presta menos atención.

El intentar retroceder y documentar un sistema, después que ha estado en operación durante algún tiempo, es más costoso que hacerlo desde la primera vez, aun considerando que el procesamiento de documentación original sea, de por si de alto costo. Este costo está determinado, en mayor grado, por el tiempo que el personal de sistemas dedica a la documentación.

Una buena documentación exige un fluxograma o diagrama de flujo inteligible que muestre los pasos y los procedimientos (por ejemplo, entradas, acceso a los datos en los dispositivos de almacenamiento secundario, cálculo, algoritmos y salidas) y el orden en el cual tienen lugar.

La documentación también debe incluir una explicación que describa en el lenguaje común, como es el procesamiento de datos. En otra forma útil se describen los datos que van a introducirse y procesarse, incluyendo su tipo y tamaño. Tal método se denomina documentación externa; es decir, que no esta incluida en el programa mismo sino que va por separado, en forma de documentación impresa en papel o en un instructivo de procedimientos.

VII.1. DOCUMENTACION INTERNA..

La documentación interna consiste en comentarios y descripciones que se insertan entre los enunciados ejecutables de un programa de computación. Explica la sucesión de los pasos del procesamiento y los objetivos de los diferentes grupos de enunciados del programa. Por ejemplo, se podrían incluir comentarios que señalen que cierto conjunto de instrucciones introducirá datos de entrada a partir del almacenamiento secundario o que una instrucción dará como resultado el cálculo del impuesto sobre ingresos basados en el pago total.

Un formato recomendado para los prólogos es el siguiente:

- Nombre del Autor:
- Fecha de compilación:
- Función(es) realizada(s):
- Algoritmos empleados:
- Autor/fecha/propósito de las modificaciones:
- Parámetros y modos:
- Condiciones de Entrada:
- Condiciones de salida:
- Variables globales:
- Efectos colaterales:
- Estructura de datos principales:
- Rutinas que invocan:
- Rutinas invocadas:

- Restricciones de tiempo:
- Manejo de excepciones:
- Suposiciones:

Los lenguajes más primitivos requerirán de prólogos más extensos. Los prólogos deben diseñarse para cada lenguaje de programación y cada proyecto. A continuación se presentan algunos principios generales básicos para los comentarios internos:

1.- Minimizar la necesidad de comentarios inmersos en el código al usar :

- Prólogos
- Estructuras de programación estructurada
- Buen estilo de programación
- Nombres descriptivos del dominio del problema para tipos definidos por el usuario, variables, parámetros formales, literales de enumeración, subprogramas, archivos, etc.
- Características de autodocumentación del lenguaje de implementación, como excepciones definidas por el usuario, tipos de datos definidos por el usuario, encapsulado de datos, etc.

2.- Adjuntar comentarios a los bloques de código que:

- Realicen manipulaciones de datos importantes
- Simulen construcciones de control estructurada que manejen instrucciones GoTo
- Realicen manejo de excepciones.

3.- Use la terminología del dominio del problema en los comentarios.

4.- Emplear líneas en blanco, delimitadores, y sangrado para realzar los comentarios.

5.- Colocar los comentarios a la extrema derecha para documentar cambios y revisiones.

6.- No manejar comentarios largos y confusos para aclarar un código oscuro y complejo. Reescríbase el código.

7.- Siempre es necesario asegurarse que el código y los comentarios correspondan uno con el otro, así como los requisitos y especificaciones de diseño.

VII.2. DOCUMENTACION EXTERNA.

La documentación es el término usado para describir todas las instrucciones, programas y aspectos narrativos, es decir, casi cualquier escrito acerca del sistema y sirve para un sin número de propósitos.

Toda la información del sistemas que nos permitirá conocer lo que hacen, como lo hace, para quién, etc., estará comprendida en la documentación de sistemas, y para que está satisfaga la característica de necesaria, oportuna y adecuada deberá estar perfectamente normalizada.

Establecer las normas de documentación por escrito es proporcionar al personal de sistemas, una herramienta que:

- Estandariza la documentación.
- Facilita el desarrollo de la misma.
- Ahorra tiempo.

Una vez que esas normas han quedado establecidas y son puestas en práctica traerá consigo una buena documentación, reportando las siguientes ventajas:

- Es una magnífica herramienta para nuevos miembros de la organización de sistemas, así como para nuevos usuarios del sistema.
- Es útil para cualquiera que tenga la responsabilidad de su mantenimiento y/o modificaciones.
- Ayuda a los analistas y diseñadores de sistemas que trabajan en áreas relativas evitando traslapes, facilitando la integración a los diferentes sistemas.
- Asegura que el sistema se opere correctamente.
- Se utilizan eficientemente los recursos de que se dispongan

De esta forma los criterios bajo los que se considera útil una documentación son los siguientes:

Disponibilidad: Cada usuario tiene que recibir, al menos un ejemplar de la documentación del sistema o programa.

Apropiada: Cada usuario debe recibir el tipo de documentación que le sea más útil.

Accesible: La documentación debe de estar bien organizada, sin saltos hacia atrás y para adelante. Debe haber una buena estructuración de la misma.

Entendible: La documentación debe ser fácil de leer y entender.

Confiable: La documentación debe de estar libre de errores, con el propósito de resolver los problemas que se presenten.

Modificable: La documentación se debe modificar fácilmente; para que cuando surjan cambios en los programas se puedan adaptar sin gran esfuerzo.

VII.2.1. Documentación del Usuario.

La documentación proporcionada a los usuarios suele ser el primer contacto que éste tiene con el sistema. Tal documentación debe proporcionar una visión inicial precisa del sistema. No es literatura comercial, así que no debe destacar en demasía las características novedosas o muy poderosas del sistema, ni debe ser poco realista acerca de las capacidades del sistema. No debe ser indispensable que el usuario lea la mayor parte de la documentación para encontrar como utilizar de forma sencilla el sistema. Por tanto, la documentación debe estructurar de forma que el usuario pueda leerla con el grado de detalle apropiado a sus necesidades. Hay al menos cinco documentos que se pueden considerar bajo el encabezamiento de documentos del usuario:

1.- Una descripción funcional sobre lo que puede hacer el sistema (manual del usuario).

2.- Un documento que explique cómo instalar el sistema y adecuarlo para configuraciones particulares del hardware (manual de Instalación).

3.- Un manual introductorio que explique, en términos sencillos, cómo iniciarse en el sistema.

4.- Un manual de referencia que describa con detalle las vèntajas del sistema disponibles para el usuario y cómo las puede usar.

5.- Una guía del operador (si se requiere un operador del sistema), que explique cómo debe reaccionar éste ante situaciones surgidas mientras el sistema se encuentra en uso.

VII.2.1.1. Manual del usuario.

Este manual tiene por objeto describir, en forma clara, los objetivos, la estructura, las capacidades y las limitaciones que tiene el sistema y, de esa manera, ayuda al usuario a comprenderlo y más tarde hacer uso eficaz del mismo.

El manual del usuario se encuentra estructurado de la siguiente manera:

a) *Portada:* En esta sección se incluye el nombre del sistema al cual pertenece el manual, el número y fecha correspondiente a la edición del mismo.

b) *Introducción:* En esta sección se describe el objetivo del manual, los capítulos que lo integran así como su forma de empleo.

c) *Indice General:* Este índice muestra como están constituídos cada uno de los capítulos que integran el manual.

d) *Descripción General del Sistema:* Este capítulo describe los objetivos del sistema al cual pertenece el manual, su estructura, los módulos que lo integran y el propósito y alcance de cada uno de ellos.

e) *Descripción General de los Archivos:* Este capítulo describe todos y cada uno de los archivos empleados por el sistema, la relación que guardan entre sí y las características de cada uno de los datos que lo integran, se estructura de la siguiente manera:

Portada del capítulo.- Esta sección contiene el nombre, número de versión y la fecha del subesquema perteneciente al sistema.

Indice por clave del archivo.- Este índice, clasificado por clave de archivo contiene está y el nombre de los archivos que intervienen en el sistema, así como el número de hoja en que se localiza su descripción dentro del capítulo.

Descripción de los archivos.- Esta sección describe la estructura de cada uno de los archivos que intervienen en el sistema, incluyendo su nombre, la descripción de su contenido, el tipo de organización, la información referente a sus llaves y los nombres de sus registros.

Diccionario de datos.- Esta sección describe el nombre, tipo de campo, la longitud, el significado y los valores que pueden adoptar cada uno de los campos que integran los archivos.

Diagrama de archivos.- Esta sección consta de uno o más diagramas que muestran la relación lógica que guardan entre sí cada uno de los archivos que intervienen en el sistema.

f) *Descripción de los Procedimientos:* Este capítulo describe cada uno de los procedimientos existentes en el sistema, teniendo presente que el manual del usuario es personalizado, es decir, sólo se incluyen aquellos procedimientos a los que el tiene acceso, su estructura es la siguiente:

Portada del capítulo.- Esta sección incluye únicamente el título y la fecha en que se edita el capítulo.

Indice estructural.- En este índice se registra cada uno de los procedimientos existentes, guardando la misma estructura que tiene el sistema. Se muestra el nombre de cada procedimiento, así como el número de hoja en que se localiza su descripción dentro del capítulo.

Indice por clave de procedimiento.- Este índice muestra los procedimientos ordenados por su clave e incluye además el nombre de éstos y el número de hoja donde se localiza por su capítulo.

Indice por nombre de procedimiento.- Este índice muestra la misma información que el anterior, pero en este caso se enlistan los procedimientos en orden alfabético.

Descripción de los procedimientos. Esta sección hace una descripción detallada de cada uno de los procedimientos incluidos en el manual del usuario, definiendo en cada uno de ellos la siguiente información:

- Clave del procedimiento.
- Nombre del procedimiento.
- Opción en el menú.
- Objetivo del procedimiento.
- Tipo de procedimiento (interactivo o remoto).
- Número de versión.
- Periodicidad de ejecución.
- Clave del procedimiento previo (si la hay).
- Parámetros utilizados por el procedimiento (si hay).
- Significado de cada uno de los parámetros utilizados por el procedimiento.
- Valores que adopta cada uno de los parámetros utilizados por el procedimiento.
- Significado de cada uno de los valores de los parámetros usados por el procedimiento.

g).-*Descripción de los Documentos Generados por el Sistema:* Este capítulo describe cada uno de los reportes generados por el sistema e incluye un ejemplo de cada uno de ellos, se encuentra estructurado de la siguiente manera:

Portada del capítulo.- Esta sección incluye el número de versión del catálogo de reportes y la fecha correspondiente a dicha versión.

Indice por clave de reporte.- Este índice muestra los reportes ordenados por su clave e incluye también el título y los procedimientos que lo integran.

Indice por título de reporte.- Este índice, muestra la misma información del anterior, pero de acuerdo a un orden alfabético de los nombres de los reportes.

Descripción de los reportes.- Esta sección describe, en forma detallada cada uno de los reportes generados por el sistema, incluyendo en cada uno de ellos la siguiente información:

- Clave del reporte.
- Título del reporte.
- Propósito del reporte.
- Clave de los procedimientos que generan los reportes.
- Periodicidad de impresión.
- Criterio de clasificación.
- Criterio de selección
- Cortes de control que se ejecutan.
- Totales obtenidos por cada corte de control.
- Distribución del reporte.
- Ejemplo del reporte.

b).- *Documentos Fuente Empleados por el Sistema:* En esta sección se anexa un ejemplo de cada uno de los documentos fuente que intervienen en el sistema, consta de una portada y el conjunto de documentos fuente.

VII.2.1.2. Manual de Instalación.

El manual de instalación debe proporcionar detalles completos sobre la manera de instalar el sistema en un ambiente particular. En primer lugar, debe contener una descripción del medio legible para la máquina en la que se proporcione el sistema: su

formato, los códigos de caracteres utilizados, cómo esta escrita la información y los archivos que componen al sistema. Después de describir la configuración mínima de hardware que se requiere para ejecutar el sistema, los archivos permanentes que deben establecerse, cómo el sistema y los archivos dependientes de la configuración que se deben modificar para adecuar al sistema a una aplicación particular.

VII.2.1.3. Manual Introductorio.

El manual introductorio debe presentar un prólogo informal que describa el uso normal del sistema. Debe explicar cómo iniciarse en el sistema y cómo se pueden utilizar sus cualidades comunes. Debe ilustrarse en forma amplia con ejemplos. El manual introductorio también debe decir al usuario del sistema cómo salir de un problema cuando las cosas funcionan mal. Los principiantes, cualquiera que sea su preparación y experiencia, cometerán inevitables errores. Es vital proporcionar información fácilmente disponible sobre cómo recuperarse de esos errores y reiniciar un trabajo útil.

VII.2.1.4. Manual de Referencia.

El manual de referencia es el documento definitivo sobre el uso del sistema. La característica más importante de un manual de referencia es que debe ser completo. Siempre que se pueda, se debe usar técnicas descriptivas formales para asegurar que se logra la plenitud y, aunque el estilo del manual de referencia no debe ser inecesariamente oscuro y pesado, es aceptable sacrificar algo de legibilidad en aras de la plenitud. Este manual puede suponer que el lector está familiarizado tanto con la descripción del sistema como con el manual introductorio. También puede suponer que el lector a utilizado en alguna forma el sistema y que comprende sus conceptos y terminologías. Además de describir con detalle las cualidades del sistema y su uso, el manual de referencia también

debe describir los informes de error generados, las situaciones en las que surgen esos errores y, si es apropiado, remitir al usuario a una descripción de la característica en la que está el error.

VII.2.1.5. Manual del operador.

En sistemas que requieren un operador, se debe proporcionar un manual a tal efecto. Este manual debe describir los mensajes que se generan en la consola del operador y cómo reaccionar ante estos mensajes. Si participa el hardware del sistema también debe explicar la tarea del operador para mantener ese hardware. Por ejemplo, debe describir cómo eliminar los fallos en la consola del operador, cómo cambiar las cintas de las impresoras, etc.

VII.3. Calidad de la Documentación.

Dependiendo del tamaño del sistema, los manuales se pueden proporcionar por separado o reunidos en uno o más volúmenes. Si se elige éste último método de presentación, cada documento se debe distinguir con claridad, de manera que los lectores puedan encontrar y usar sin complicaciones los documentos que necesitan. Esta distinción se debe hacer para separar las secciones y proporcionar índices alfabéticos, o puede lograrse mediante la impresión de las distintas partes del manual en papel de diferentes colores.

Además de manuales, puede ser apropiado proporcionar a los usuarios otra documentación de uso fácil. Por ejemplo, una tarjeta de referencia rápida que liste las ventajas disponibles del sistema y cómo usarlas, en particular es muy conveniente para usuarios experimentados. Los sistemas de ayuda en línea, que contienen información breve

del sistema, son otra característica que evita al usuario perder tiempo consultando manuales.

Algunas organizaciones consideran que producir documentación para el usuario no debe ser tarea del ingeniero de software, si no que deben emplearse escritores técnicos profesionales para producir tal documentación, utilizando la información proporcionada por los ingenieros responsables de la construcción del sistema. Este enfoque tiene cierto mérito, pues deja libre al personal de software para hacer su trabajo principal: construir software. Sin embargo, tiene el inconveniente de que la comunicación entre los escritores y los ingenieros de software puede consumir casi tanto tiempo como escribir la documentación, de modo que, en la práctica, el uso de escritores técnicos puede no ser rentable.

Es una situación lamentable el que, en la ingeniería de software, la mayor parte de la documentación de los sistemas de cómputo esté mal escrita, sea difícil de entender y se encuentre desactualizada o incompleta. Con algunas honrosas excepciones, se ha puesto poca atención a la producción de documentos de sistemas que tengan una prosa técnica bien escrita. Suele considerarse que la documentación es algo que se debe terminarse con rapidez después del trabajo más interesante de desarrollar el software.

De hecho, la calidad de la documentación es tan importante como la calidad del programa. Sin información sobre la utilización del sistema o sobre su comprensión, la utilidad del sistema queda mermada. Producir buenos documentos no es fácil ni barato; el proceso es casi tan difícil como producir buenos programas. Por tanto, se proporcionarán algunas sugerencias sobre la producción de documentos de sistemas de software de buena calidad.

Un procedimiento estándar para producir, revisar y acomodar la documentación es ayuda importante para el control de la calidad de la documentación. Nunca será excesivo destacar la importancia de los mecanismos para el control de la calidad en la producción

de documentos. Una documentación pobre quizá confunda más de lo que ayude al usuario, con la consecuencia de que es probable que se haga un uso mínimo de ella.

Los estándares para la documentación tiene que describir con exactitud lo que debe incluir la documentación y las notaciones a utilizar en ella. Dentro de una organización, es útil establecer un "formato estándar de la casa" y solicitar que todos los documentos se ajusten a él. Dicho estandar puede incluir la descripción de un formato para la cubierta frontal para ser adoptado en todos los documentos, las conversiones para la numeración y anotación de páginas, los métodos para hacer referencia a otros documentos y la numeración de los títulos y subtítulos.

La numeración de los títulos y subtítulos puede parecer un asunto trivial. Sin embargo, cuando se produce una gran cantidad de documentos separados que hacen referencia unos a otros, es vital un esquema de numeración consistente. Los documentos se deben subdividir usando exactamente el mismo sistema, de modo que cada una de las secciones correspondientes de cada documento se refieran a la misma entidad. Se debe evitar, en lo posible, hacer referencias cruzadas de manera explícita y, cuando resulte inevitable, la referencia se debe hacer por título y número de sección.

VII.3.1. Estilo de Redacción.

Aunque los estándares y la evaluación de la calidad son esenciales si se quiere producir una buena documentación, el factor más importante en ese sentido es la habilidad del redactor para construir una prosa técnica, clara y concisa. Dicho en pocas palabras, la buena documentación requiere una buena redacción. En un libro de esta naturaleza, dedicado a un tema técnico, puede parecer presuntuoso incluir notas sobre estilo de redacción. Sin embargo, lamentablemente es un hecho que existen personas que participan en la

producción de software que tienen grandes dificultades para elaborar una documentación bien escrita, clara y concisa.

Escribir documentos bien no es fácil ni constituye un proceso de una sola etapa. Se debe redactar, leer, criticar y volver a escribir, repitiendo el proceso hasta que se produzca un documento satisfactorio. Como en otros aspectos de la ingeniería de software, es imposible presentar un conjunto de reglas que determinen en forma exacta cómo realizar esta actividad. La redacción técnica es un arte más que una ciencia y sólo se pueden dar unas líneas generales acerca de cómo escribir bien. Estos principios generales, dispuestos en un estilo que se puede usar en los manuales de instrucción del software, son:

- *Utilizar formas gramaticales activas en vez de pasivas.* Por ejemplo, es mejor decir "verá un cursor intermitente en la parte superior izquierda de la pantalla" que "un cursor intermitente será visto en la parte superior izquierda de la pantalla".

- *No emplear frases largas que presenten varios hechos distintos.* Es mucho mejor usar varias oraciones cortas. Así cada oración se puede asimilar por sí sola. El lector no necesita retener varias partes de información a la vez para comprender la oración completa.

- *No hacer referencia a la información sólo con un número de referencia.* En lugar de esto, se debe dar el número de referencia y recordar al lector lo que ésta cubre. Por ejemplo, en vez de decir "en la sección 1.1.1...", se debe decir "en la sección 1.1.1, que describe el proceso de evolución del software,...".

- *Detallar los hechos siempre que sea posible.* Por lo general, es más claro presentar hechos en una lista que en una frase. Habría sido difícil leer este conjunto de principios generales si no se hubiese detallado.

- *Si determina que la descripción es compleja, debe repetirse.* A menudo es buena idea presentar dos o más descripciones de la misma cosa. Si el lector no puede comprender una descripción, puede serle de utilidad tener la misma situación expresada de distinta manera.

- *Ser concreto.* Si se puede decir algo en cinco palabras, debe hacerse así, en vez de usar diez palabras para que la descripción parezca más profunda. Lo importante no es la cantidad de documentación sino su calidad.

- *Ser preciso y definir los términos utilizados.* La terminología de la computación es muy fluida, y muchos términos tienen más de un significado. Por tanto, si se usan tales términos (por ejemplo, módulo o proceso), debe asegurarse que su definición es clara. Si es necesario definir varias palabras, o si se describe para lectores con poco o ningún conocimiento de la terminología de computación, se debe proporcionar un glosario junto con el documento. Este glosario debe contener las definiciones de todos los términos que quizá el lector no comprenda del todo.

- *Utilizar párrafos cortos.* Como regla general, ningún párrafo se debe componer de más de siete frases. La capacidad de retener información inmediata es limitada, por lo que, mediante párrafos cortos, todos los conceptos del párrafo se pueden retener en la memoria a corto plazo.

- *Utilizar títulos y subtítulos.* Hay que asegurarse siempre de utilizar una convención de numeración consistente.

- *Utilizar construcciones gramaticales y ortografía correcta.* El uso excesivo de infinitivos y las faltas de ortografía irritan a muchos lectores y reduce la credibilidad del escritor.

Un mecanismo para examinar y mejorar los documentos es establecer inspecciones de documentos. Estas se utilizarán de la misma manera que las inspecciones del código para la determinación de errores en los programas. Durante la inspección de un documento se critica el texto, se señalan las omisiones y se hacen sugerencias para mejorarlo. En este último aspecto difiere de la inspección de código, que no es otra cosa que un simple mecanismo para encontrar errores y no para corregirlos.

Además de una crítica personal, también se pueden utilizar instrumentos de software cuya función es leer texto y encontrar fallas gramaticales o usos inapropiados de las palabras. Estos instrumentos también pueden indicar en que parte las oraciones o los párrafos son demasiado largos y en donde se utilizan formas gramaticales pasivas en vez de activas.

CAPITULO VIII

PRUEBA Y DEPURACION DE L SISTEMA

INTRODUCCION.

Cuando se prueba un programa se desea agregarle algún valor. Agregar valor significa aumentar su calidad o su confiabilidad, lo que, a su vez, significa encontrar y eliminar errores.

Prueba es el proceso de ejecutar un programa con el fin de encontrar errores.

La prueba de programas puede se considerada propiamente como el proceso destructivo de tratar de encontrar errores. Un caso exitoso de prueba es el que conduce un progreso en esta dirección, causando una falla del programa, se desea finalmente hacer una prueba de programas para establecer algún grado de confiabilidad en que el programa hace lo que se supone que debe hacer y que no hace lo que no se supone que debe hacer. Si alguien afirma " mi programa es perfecto", entonces es la mejor manera de establecer cierta confianza en esta declaración es tratar de refutarla, es decir tratar de encontrar imperfecciones, más bien que confirmar que el programa funciona adecuadamente para algún conjunto de datos de entrada.

Es bién conocido que cuando antes se encuentre los errores, más bajo será el costo de corregirlos y mayor será la probabilidad de hacerlo bien, los programadores experimentan al parecer un cambio psicológico cuando comienzan las pruebas basadas en computadora. Las presiones internas parecen crecer rápidamente y hay una tendencia a querer "arreglar esta dichosa falla lo antes posible". Por causa de estas presiones, los programadores tienden a cometer más equivocaciones cuando corrigen errores encontrados mediante pruebas efectuadas con computadora que cuando se trata de errores encontrados antes.

VIII.1. CARACTERISTICAS DE LA PRUEBA.

Para que una prueba sea confiable se deben seguir los siguientes pasos:

1.- Un programador debe evitar probar sus propios programas.

2.- Una empresa de programación no debería probar sus programas.

3.- Inspeccionar concienzudamente el resultado de cada prueba.

4.- Los casos de prueba deben ser escritos tanto para condiciones de entrada inválidas e inesperadas como para condiciones válidas y esperadas.

5.- Examinar un programa para comprobar que no hace lo que se supone que debe hacer es solo la mitad del programa. La otra mitad consiste en ver si el programa hace lo que no se supone que debe hacer.

6.- Evite los casos de prueba desechables a menos que el programa sea verdaderamente un programa desechable.

7.- No planear un esfuerzo de prueba con la suposición tácita de que no se encontrarán errores.

8.- La posibilidad de encontrar errores adicionales es una sección del programa es proporcional al número de errores ya encontrados en esa misma sección.

9.- Las pruebas constituyen una tarea altamente creativa y son un desafío intelectual.

10.- Prueba en el proceso de ejecutar un programa con la intención de encontrar errores.

11.- Un buen caso de prueba es el que tiene una probabilidad elevada de encontrar un error aún encubierto.

VIII.2. PASOS EN LA PRUEBA DE LOS SISTEMAS DE PROGRAMACION.

La planificación es una parte decisiva de la organización del proceso de prueba. Los componentes de un buen plan son:

1.- Objetivos. Deben definirse los objetivos de cada fase de la prueba.

2.- Criterios de terminación. Deben especificarse los criterios para determinar cuándo se puede considerar terminada cada fase de la prueba.

3.- Cronograma. Se necesita fijar los tiempos necesarios para cada fase y ejecutar los casos de prueba.

4.- Responsabilidades. Para cada fase será establecido quién diseñará, escribirá, ejecutará y verificará los casos de prueba, se designarán las personas que han de corregir los errores que fueren encontrados.

5.- Bibliotecas de casos de prueba y normas. En proyectos grandes son necesarios ciertos métodos sistemáticos para identificar, escribir y almacenar casos de prueba.

6.- Herramientas. Debe establecerse cuáles serán las herramientas de prueba que se han de emplear en el proyecto.

7.- Tiempo de máquina. Habrá que planificar el tiempo de computación que se necesitará en cada fase del proyecto de prueba.

8.- Configuración del equipo. Si se necesitan configuraciones especiales de equipo o se requiere algún periférico en particular, será necesaria una planificación que describa los requerimientos.

9.- Integración. Una parte del plan de prueba es una definición de cómo se armará el programa. Un sistema que contiene subsistemas grandes o programas podría también ser armado incrementalmente.

10.- Métodos de seguimiento. Se deben identificar los medios para seguir los múltiples aspectos del progreso de las pruebas.

VIII.3. GENERADORES DE DATOS DE PRUEBA.

Existen dos fuentes de datos de prueba fundamentalmente diferentes: los datos reales y los artificiales. Ambos tienen ventajas y desventajas distintas para quien hace la prueba.

VIII.3.1. Utilización de datos reales de prueba.

Los datos de prueba son los que se extraen de los archivos de la empresa. Después de que se ha construido en forma parcial un sistema, los programadores o analistas con frecuencia piden a los usuarios que tecleen un conjunto de datos como si fuera parte de sus actividades normales. Por ejemplo, en un sistema de contabilidad general pueden

pedirle a alguien del departamento de contabilidad que introduzca los números del catálogo de cuentas y un conjunto de saldos de cuentas, junto con las transacciones que afecten a estas cuentas. Entonces el personal del departamento de sistemas utiliza estos datos como una forma de probar en forma parcial el sistema. En otras palabras, los programadores o analistas extraerán un conjunto de datos reales de los archivos y los introducen ellos mismos.

Es difícil obtener datos reales en cantidades suficientes para llevar a cabo una prueba extensiva y, aunque son datos reales que mostrarán como se desempeñará el sistema para los requerimientos normales de procesamiento, suponiendo que los datos reales introducidos son de hecho normales, generalmente no probará todas las combinaciones o formatos que pueden introducirse al sistema. El sesgo hacia valores normales entonces no proporciona una prueba verdadera de sistemas y de hecho ignora los casos que probablemente ocasionan la falla de los sistemas.

VIII.3.2. Utilización de datos artificiales de prueba.

Los datos artificiales para prueba se crean únicamente para propósitos de prueba, dado que se pueden generar para probar todas las combinaciones de formatos y valores; en otras palabras, los datos artificiales, que se pueden preparar rápidamente por un programa de utilería de generación de datos en el departamento de sistemas de información, hace posible la prueba de todas las rutas lógicas y de control a través del programa.

Los programas de prueba más efectivos utilizan datos artificiales para prueba generados por personas diferentes a las que escribieron los programas. A menudo un grupo independiente hace las pruebas formulando un plan de prueba al utilizar las especificaciones de los sistemas.

VIII.4. PRUEBA DE UNIDADES Y PRUEBA DE INTEGRACION.

VIII.4.1. Prueba de Unidades.

Las pruebas de unidad comprenden el conjunto de pruebas efectuadas por un programador individual, antes de la integración de la unidad en un sistema más grande. La situación se ilustra como sigue:

Codificación y depuración - Una unidad de programa suele ser lo suficientemente pequeña como para que el programador que la desarrolló pueda probarla con minuciosidad, esta prueba debe ser mucho más minuciosa que el exámen al que se someterá cuando la unidad se integre en el producto de software en desarrollo. Hay cuatro categorías de pruebas que, por lo común, efectuará un programador a una unidad de programa:

Pruebas funcionales.- Estos implican ejercitar el código con valores nominales de entrada para los cuales se conocen los resultados esperados, además de valores límites y los valores especiales.

Pruebas de desempeño.- Determinan la cantidad de tiempo de ejecución empleado en varias partes de la unidad, la eficiencia global del programa, el tiempo de respuesta y la utilización de dispositivos por la unidad de programa. Las pruebas de desempeño son más productivas en los niveles de subsistema y de sistema.

Pruebas de tensión.- Son aquellas diseñadas para romper, en forma intencional, la unidad. Se puede aprender mucho acerca de la resistencia y limitaciones de un programa examinando cómo se rompe una unidad de programa.

Pruebas de estructura.- Las pruebas de estructura se relacionan con ejercitar la lógica interna de un programa y recorrer rutas de ejecución particulares.

Algunos autores se refieren colectivamente a las pruebas funcional, de desempeño y de tensión como pruebas de "caja negra", mientras que a las estructuras las denominan pruebas de "caja blanca" o "de cristal".

VIII.4.2. Pruebas de Integración.

La integración ascendente es la estrategia tradicional empleada para integrar los componentes de un sistema de software en un todo funcionando. La integración ascendente consiste en pruebas de unidad, seguidas por pruebas de subsistemas y luego por pruebas de sistema completo. Las primeras tienen el objetivo de descubrir errores en los módulos individuales del sistema. Estos módulos se prueban aislados unos de otros en un ambiente artificial llamado "arreo de prueba", el arreo está formado por los programas conductores y los datos necesarios para ejercitar los módulos. El propósito principal de las pruebas de subsistema es verificar la operación de las interfaces entre módulos en el subsistema. Se deben probar tanto las interfaces de control como la de datos. Las pruebas del sistema se relacionan con sutilezas en las interfaces, la lógica de decisión, el flujo de control, los procedimientos de recuperación, la eficiencia global, la capacidad y las características de tiempo del sistema entero. Se requiere una escrupulosa planeación de las pruebas para determinar la extesión y la naturaleza de las pruebas del sistema que se van a realizar y establecer los criterios para la evaluación de los resultados.

Las pruebas ascendentes tienen la desventaja de que necesitan escribir y depurar arreos de pruebas para los módulos y subsistemas, además, el nivel de complejidad adquirido al combinar módulos y subsistemas en unidades cada vez más grandes. El caso extremo de complejidad resulta que cuando en cada módulo se efectúan pruebas de unidad por separado y después todos los módulos se ligan y ejecutan en una sola corrida de integración.

La integración descendente empieza con la rutina principal y una o dos rutinas inmediatamente subordinadas en la estructura del sistema. Después de que este esqueleto de alto nivel ha sido probado con detenimiento, se convierte en el arreo de prueba para sus subrutinas inmediatamente subordinadas. La integración de alto nivel requiere el uso

de troncos de programa para simular el efecto de las rutinas de más bajo nivel que son llamadas por las rutinas en la prueba.

Aunque pudiera parecer que la integración descendente siempre es preferible, se presentan muchas ocasiones en las que es imposible apegarse a una estrategia de codificación e integración estrictamente descendente.

La integración emparedado es predominantemente descendente, pero las técnicas ascendentes se usan en algunos módulos y subsistemas. Esta mezcla mitiga mucho de los problemas encontrados en las pruebas descendentes puras y, además retiene las ventajas de la integración descendente al nivel de subsistema y del sistema.

VIII.5. PRUEBAS DE VALIDACION.

Tras la culminación de la prueba de integración, el sofware está completamente ensamblado como un paquete; se han encontrado y corregido los errores de interfaces, y debe comenzar una serie final de pruebas de software. La prueba de validación. La validación puede ser definida de muchas formas, pero una simple indicación es que la validación se logra cuando el software funciona de acuerdo con las expectativas razonables del cliente.

Las espectativas razonables están definidas en la Especificación de Requerimientos del Software un documento que describe todos los atributos del software que son visibles al usuario. La especificación contiene una sección denominada Criterios de Validación. La información contenida en esa sección forma la base de la aproximación a la prueba de validación.

VIII.5.1. Criterios para la prueba de validación.

La validación del software se consigue mediante una serie de pruebas de la caja negra que demuestran la conformidad con los requerimientos. Un plan de prueba traza las pruebas que se han de llevar a cabo, y un procedimiento de pruebas define los casos de prueba específicos que serán usados para demostrar la conformidad con los requerimientos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requerimientos funcionales, que se alcanzan todos los requerimientos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requerimientos. (p. ej., portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento).

Una vez que se procede con cada caso de prueba de validación, puede darse una de dos condiciones: 1) las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptables, 0 2) se descubre una desviación de las especificaciones y se crea una lista de deficiencias. Las desviaciones o errores descubiertos en esta fase del proyecto raramente se pueden corregir antes de terminar el plan. A menudo es necesario negociar con el cliente un método para resolver las deficiencias.

Es virtualmente imposible que un encargado del desarrollo del software pueda prever cómo un cliente usará realmente un programa. Se pueden interpretar mal las instrucciones de uso, se puden usar regularmente extrañas combinaciones de datos y una salida que puede estar clara para el que realiza la prueba y puede resultar ininteligible para un usuario normal.

Cuando se construye software a medida para un cliente, se lleva a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requerimientos. Llevado a cabo por el usuario final en lugar del equipo de desarrollo, una prueba de aceptación puede ir desde un informal "prueba de prueba" hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, la prueba de aceptación puede tener

lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema.

Si el software se desarrolla como un producto a ser usado por muchos clientes, no es práctico realizar pruebas de aceptación formales para cada uno de ellos. La mayoría de los constructores de productos de software llevan a cabo un proceso denominado prueba alfa y beta para descubrir errores que parezcan que sólo el usuario final puede descubrir.

La prueba alfa es conducida por un cliente en el lugar de desarrollo. Se usa el software en forma natural, con el encargado del desarrollo "mirando por encima del hombro" del usuario y registrando errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.

La prueba beta se lleva a cabo en uno o más lugares de clientes por los usuarios finales del software. A diferencia de la prueba alfa, el encargado del desarrollo normalmente no está presente. Así la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el equipo de desarrollo. El cliente registra todos los problemas (reales e imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al equipo de desarrollo. Como resultado de los problemas anotados durante la prueba beta, el equipo de desarrollo del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la base de clientes.

VIII.6. PRUEBAS DE VOLUMEN.

Este tipo de prueba somete al programa a la acción de grandes volúmenes de datos. Si un programa está diseñado para manejar archivos que abarcan múltiples volúmenes, entonces se generán suficientes datos para obligar al programa a pasar de un volúmen a otro. En otras palabras el propósito de la prueba de volúmen es mostrar que el programa no puede manejar los volúmenes de datos especificados en los objetivos.

La prueba de volúmen es una prueba costosa, tanto en tiempo de máquina como en personal, se debe tratar de no exceder los límites. Sin embargo, todo programa debería ser expuesto, al menos, a algunas pruebas de volúmen.

VIII.7. SIMULACION DEL SISTEMA.

Los simuladores de ambiente, en ocasiones, se utilizan durante las pruebas de integración y aceptación para aparentar el ambiente de operación en el que funcionará el software. Los simuladores se usan en situaciones en que la operación del ambiente real es imprácticа; como, por ejemplo, el desarrollo del software para máquinas no existentes, simuladores de entradas de tiempo real para un sistema no existente o de costosa operación y situaciones donde las pruebas "en vivo" son imposibles. Ejemplode simuladores de ambiente son PRIM (GAL75) para emular maquinas inexistentes y el simulador del programa de vuelo saturno aparenta pruebas de vuelos en vivo (JAC70).

CAPITULO IX

INSTALACION Y APROBACION DEL SISTEMA.

INTRODUCCION.

En este capítulo se verán algunos puntos importantes en toda instalación de un sistema, dependiendo de la importancia que tenga en la empresa o si es un sistema que reemplaza a una versión vieja o es nueva.

Otro punto muy importante que se trata en este capítulo es el de la capacitación del personal que va a operar el sistema y el mantenimiento, es por eso que se mencionan algunas de las ventajas y desventajas que se tienen a los diferentes tipos de capacitación.

IX.1. IMPLANTACION DEL SISTEMA

Una vez aprobado el sistema propuesto, se deberá diseñar con detalle cada paso, tomando en consideración las observaciones que se hayan hecho al sistema durante sus presentaciones.

Otros aspectos por considerar son: la preparación de los elementos humanos que intervendrán en cada una de las actividades que requiere el proyecto, los recursos con que se decidió implantar el sistema, así como las características del equipo por utilizar, para aprovecharlo mejor. La implantación es la fase donde se pone en práctica el concepto teórico de un sistema, es la más objetiva del desarrollo; es la culminación de actividades como plantación, análisis, diseño y programación del nuevo sistema. Es la parte en la que el analista debe mostrar su habilidad y conocimientos para lograr el máximo aprovechamiento y efectividad de todos los recursos humanos y materiales, mediante una coordinación precisa y cuidadosa.

Lo anterior requiere del conocimiento complejo del sistema, pero esto no quiere decir que él tenga que hacer todo, sin que mediante bases de plantación y estructuras organizacionales debe delegar en otras personas ciertas funciones de detalle.

Entre las funciones de la implantación por realizar están las siguientes:

Diagramación general

Este diagrama especificará en forma muy general el recorrido de la información, indicando las áreas internas y dependencias externas que intervendrán en el sistema.

Diagrama particular de procedimientos manuales

Este diagrama es el desglose específico de las actividades a realizar por cada área usuaria, primordialmente en lo que se refiere a captación de información (codificación y realimentación de información) y uso de cada uno de los resultados por obtener en el sistema: Por otro lado, el diagrama también comprende las actividades de conversión de información fuente a medio procesable.

Diagrama de procedimientos electrónicos

Es la representación gráfica del flujo y proceso de la información, en un computador para obtener los resultados previstos en el sistema. El diseño de este procedimiento dependerá por un lado del tipo y configuración del equipo de la instalación y, por el otro, de la calidad del personal con que se cuenta en programación. Por último, se debe considerar la no saturación del equipo con un solo proceso.

IX.2. Definición de programas

Al llegar a este punto, el diseñador ha invertido meses de tiempo y esfuerzo en la investigación y diseño; sin embargo, durante el diseño cualquier cambio, cualquier factor descuidado previamente; por lo general no causa retrasos o riesgos.

El analista es el responsable de cuidar que los programas desarrollados estén de acuerdo con las especificaciones de los procesos diseñados.

Es importante saber que es fácil hacer cambios en el sistema antes de programarlo, pero una vez hecho esto, cualquier cambio se dificulta; esto significa que el sistema propuesto debe estar correcto y completo antes de pasar las definiciones a la fase de programación.

Para llevar a cabo las definiciones de cada uno de los programas, es conveniente que el analista se apegue a los siguientes puntos:

Elementos para la definición de un programa

— *Identificación.* Estará formada por los siguientes datos:

- Nombre del sistema y subsistema.
- Nombre del analista.
- Clave del programa.
- Datos cronológicos.

— *Entradas.* Aquí se proporciona la información genérica con las características de los archivos de entrada, tales como:

- Nombre externo del archivo.
- Posiciones por registro.
- Factor de bloque o registros por bloque.
- Dispositivo o dispositivos.
- Posición por la que está clasificado (si está clasificado).
- Organización.
- Llaves de acceso.

— Método de acceso

— Tipo de etiquetas.

— Nombre interno.

- Número de volumen de dispositivo.

- *Salidas.* Su contenido es similar a lo expuesto en el párrafo de entradas, salvo para resultados por impresora, en cuyo caso deberá anotarse la clave de la "forma de papel" que se utilizará, número de tantos, si la impresión es a sextos u octavos, cinta de control, carro normal o especial, o en su defecto, parámetros de control de saltos de impresión.

- *Proceso.* Esta parte de la definición de un programa es la que normalmente presenta una mayor dificultad para su realización y entendimiento; esto se debe principalmente al hecho de pretender decir cómo se quiere hacer, y por consiguiente, no se atiende lo que se desea que el programa haga. La descripción del proceso debe ser lo más simple posible, atendiendo con precisión los resultados que se desean.

- *Procesos de conversión y depuración.* En este tipo de procesos es frecuente que sólo se depuren los campos considerados como información básica, descuidando el resto de la información. Por consiguiente, se sugiere que este tipo de programas hagan una depuración más exhaustiva de todos y cada uno de los campos que integran un registro de información.

- *Especificación de campos por procesar.* Cuando se haga referencia a los campos sujetos al proceso, deberán corresponder, tanto la descripción del campo como las posiciones que ocupa dentro del registro, a las especificaciones en el formato de registro.

- *Procesos de datos clasificados.* Cuando la información se presenta clasificada, con el propósito de producir uno o más cortes, o bien acumulaciones de información que corresponden a la misma llave de clasificación, es importante que se den los criterios para que, al verificar la secuencia, el programa tenga la opción de tratamiento en caso de inconsistencia. Esta puede ser fuera de secuencia, registros duplicados, o especificar que el archivo no está clasificado.

- *Cruces de archivos para actualización y/o complementos de información.* Aparte de lo expuesto en el párrafo relativo a información clasificada en este tipo de procesos, es de gran importancia dar los criterios para cuando la información no encuentra correspondencia en uno o más de los archivos objetos del proceso. Nunca se debe presuponer que toda la información se encuentra correspondiente; o que un archivo se termina primero que otro, o bien que el programador tiene los antecedentes para dar un tratamiento adecuado a esta información, ya que sólo producirá resultados inadecuados.

- *Rutinas de cálculo.* Para toda rutina de cálculo el analista debe proveer, dentro de la definición, las fórmulas matemáticas mediante las cuales se llega al resultado deseado.

Lo anterior es más claro y preciso que pretender describir en forma literaria dichas fórmulas.

- *Programa con dos o más opciones.* En estos casos deberá poner atención en no hacer confusa la definición, procurando dejar independiente cada opción y dando marcada preferencia a la manera el que el programa optará por alguna de sus opciones; en otras palabras, deberá especificarse el método de selección de la opción.

- *Cifras de control.* Esta parte de la definición requiere mucha atención por parte del analista; se debe cuidar especialmente que las cifras de control cubran su principal objetivo: controlar los resultados del programa. Esta valiosa herramienta por lo general no se usa en forma adecuada, resultando a veces muy problemático o imposible cuadrar dichas cifras de control.

IX.3. MANTENIMIENTO.

Una vez implantado un sistema, es necesario darle mantenimiento para cerciorarse de su buen funcionamiento y adaptación a las necesidades del momento, ya sean del sistema electrónico o de recursos técnicos. A un sistema se le debe dar mantenimiento básicamente por actualizaciones del sistema o por racionalización del mismo.

IX.3.1 Actualizaciones en la variación de procesos.

Las variaciones de procesos pueden ser:

Entradas. Las variaciones en las entradas de un sistema pueden estar relacionadas con el cambio del documento fuente y/o con los recursos de captación, generándose una variación en el sistema; por lo que es necesario actualizarlo, ya sea parcial o totalmente.

Salidas. Las variaciones en las salidas de un sistema pueden estar relacionadas con las variaciones de entrada y/o con las variaciones de operación. Es decir, si la creación de un nuevo resultado, o la modificación a otro ya existente, afectan las entradas de un sistema, lógicamente afectarán las salidas del mismo, y en consecuencia su operación, por lo que es necesario actualizar el sistema, ya sea parcial o profundamente.

Operación. Las variaciones de operación son producto de las variaciones de entradas y/o salidas de un sistema, o bien por fallas de proceso; por lo que es necesario actualizarlo. Habrá variaciones de operación en donde no será necesaria la actualización del sistema, únicamente se hará un ajuste en la fase afectada.

IX.3.2. Actualizaciones en la variación de recursos.

Las variaciones de recursos de un sistema pueden ser:

Humanos. En la implantación de un sistema se deben indicar los recursos humanos necesarios para el manejo del mismo. La variación de este recurso puede repercutir en el buen funcionamiento del sistema; por lo cual será necesario realizar los ajustes correspondientes para que estas variaciones no interfieran en su funcionamiento.

Materiales. Los recursos materiales de que dispone un sistema están estrechamente ligados con los recursos materiales de que disponen las unidades que intervienen en él. Al cambiar el tipo de recursos materiales de las unidades, un sistema podrá variar a fin de adaptar el sistema a estos nuevos recursos, sin que esto afecte los objetivos originales.

IX.4. CAPACITACION

Incluso los sistemas bien diseñados y técnicamente elegantes pueden tener éxito o fallar debido a la forma en que se opera y se utilizan, por lo tanto, la calidad de la capacitación del personal involucrado con el sistema en varias de sus facetas, ayuda o dificulta y puede obstaculizar por entero el éxito de la puesta en marcha de un sistema de información. Las personas que trabajan con el sistema o que se verán afectadas por este deben conocer con detalle las funciones que desempeñaran, como utilizaran el sistema y lo que esta hará o no.

IX.4.1. Capacitación de operadores de sistemas.

Muchos sistemas dependen del personal del centro de computo, quien tiene la responsabilidad de mantener el equipo en buenas condiciones, así como de proporcionar el apoyo necesario. Su capacitación debe garantizar que esta en condiciones para manejar todas las operaciones posibles, tanto las de rutina como las extraordinarias.

Si el sistema requiere la instalación de equipo nuevo, por ejemplo, de un nuevo sistema de computo, de terminales especiales o de equipo diferente para entrada de datos, la capacitación del operador deberá incluir aspectos fundamentales, como la manera en que se debe encender el equipo o la forma de operarlo.

IX.4.2. Capacitación a usuarios.

La capacitación de usuario puede incluir el empleo de equipo, en especial, por ejemplo, en el caso de que se emplee una microcomputadora y el individuo sea simultáneamente operador y usuario

La capacitación del usuario también debe instruirlo en la solución de problemas dentro del sistema, determinado si cuando surge uno se origina en el equipo, por el SOFTWARE o en alguna la acción que se haya llevado acabo al operar el sistema. Incluir una guia de problemas comunes en la documentación del sistema proporcionara una útil referencia durante mucho tiempo, después de que concluya el periodo de capacitación.

La mayor parte de la capacitación de los usuarios radica en la operación del sistema mismo.

Las actividades del manejo de datos que reciben la máxima atención dentro de la capacitación de usuarios son la entrada de nuevos datos, la edición de datos, la formulación de consultas (la búsqueda de registros específicos) o la obtención de respuestas a las preguntas y borrar registros de datos.

De vez en cuando los usuarios deberan preparar discos, cargar papel o cambiar las cintas en las impresoras. Ningún programa de capacitación estará completo si no dedica algún tiempo a las actividades de mantenimiento de sistemas. Si un sistema de microcomputadoras a la entrada de datos emplea discos, los usuarios deberan recibir instrucciones al respecto al método de formateo y probar discos

Como lo demuestra el análisis anterior, existen dos aspectos para la capacitación del usuario: capacitación familiaridad con el sistema de procesamiento (es decir, el SOFTWARE que acepta los datos, los procesa y produce los resultados).

IX.4.3. Métodos de capacitación.

la capacitación de los operarios y usuarios se puede lograr mediante diferentes formas. Las actividades se pueden llevar acabo en las mismas instalaciones de venta de equipos,

en lugares alquilados, por ejemplo en hoteles, universidad o en instalaciones que pertenezcan a la compañía donde labora los resultados.

IX.4.4. Capacitación por parte de los proveedores y en los locales de servicios.

Con frecuencia el mejor instructor para capacitar al personal sobre el equipo proviene del vendedor que lo proporciona.

IBM, por ejemplo, ofrece cursos complementarios de dos o tres días a los compradores de muchas computadoras grandes y pequeñas. los cursos conducidos por capacitadores experimentados y por personal de ventas cubren todos los aspectos del empleo de equipo, desde como encenderlo y apagarlo, hasta el almacenamiento y eliminación de datos, incluyendo el manejo de las funciones diferentes. Este entrenamiento es practico, de manera que los participantes emplean, de hecho, el sistema en presencia de los capacitadores.

IX.4.5. Capacitación dentro de la compañía.

La ventaja de ofrecer capacitación para manejar el sistema dentro de la compañia es que la instrucción se puede adecuar a las necesidades de la empresa donde se ofrece, y se puede centrar en los procedimientos especiales que se emplean, en el crecimiento que se ha planeado y en los problemas que surgen.

También existen desventajas; el sólo hecho de que los empleados se encuentren en su lugar común de trabajo constituye una distracción, ya que las llamadas telefónicas y las emergencias pueden interrumpir las sesiones de capacitación.

Los manuales de capacitación resultan aceptables para fines de familiaridad, pero la experiencia de utilizar de manera práctica el equipo, cometer errores. o encontrar circunstancias inesperadas son la mejor y más duradera forma para aprender.

IX.5. CONVERSION.

La conversión es el proceso de cambio del sistema antiguo al nuevo.

IX.5.1. Métodos de conversión.

Existen cuatro métodos para manejar la conversión de sistemas. Cada método debe considerarse de acuerdo con las oportunidades que ofrece y los problemas que pudiera originar.

IX.5.2. Sistemas paralelos.

El método más seguro de conversión de un sistema viejo a uno nuevo radica en operar ambos sistemas en forma paralela, lo que significa que los usuarios continúan operando el viejo sistema de la manera acostumbrada, pero empiezan también a emplear el nuevo sistema. Este método constituye el enfoque de conversión más seguro, dado que garantiza que en caso de que aparezcan problemas durante el empleo del nuevo sistema, como errores de procesamiento o incapacidad para manejar determinados tipos de transacciones, la empresa aún podrá valerse del viejo sistema sin pérdida de tiempo, ingresos o servicios.

Las desventajas del enfoque de sistemas paralelos son significativas. En primer lugar, los costos del sistema se duplican, ya que existe ahora dos conjuntos de costos de sistemas. En algunos casos, es necesario contratar personal eventual que ayude a la operación de

ambos sistemas en paralelo. En segundo lugar, el hecho de que los usuarios saben que pueden apoyarse en los viejos sistemas puede convertirse en una desventaja si existe resistencia potencial para el cambio o si los usuarios prefieren el viejo sistema.

IX.5.3. Cambio directo.

El método de cambio directo convierte el sistema viejo al nuevo de manera repentina, en ocasiones en el lapso de un fin de semana o incluso de un día para otro. El sistema viejo se emplea hasta el día de conversión planeado y entonces se reemplaza por el sistema nuevo.

La ventaja de no tener un sistema de apoyo se puede convertir en una desventaja si surgen problemas serios con el sistema nuevo. En algunos casos las empresas pueden incluso, detener las operaciones cuando surgen problemas y ésto se puede corregir.

IX.5.4. Enfoque piloto.

Cuando los sistema son nuevos implican también técnicas nuevas o cambios drásticos en el desempeño de la empresa se prefiere el enfoque piloto con frecuencia.

En este método una versión práctica del sistema se pone en marcha en una parte de la empresa; por ejemplo, una sóla área o departamento de trabajo. Los usuarios de esta área saben por lo general que estan llevando a cabo una prueba piloto de un nuevo sistema y que pueden practicarse cambios para mejorarlo.

Cuando se calcula que el sistema está completo, se instala en toda la compañia ya sea de una sola vez (cambio directo) o en forma gradual (por etapas).

IX.5.5. Método por etapas.

El método por etapas se emplea cuando no es posible instalar un sistema nuevo en toda la compañía de manera simultánea. La conversión de archivo, la capacitación del personal o la llegada del equipo pueden forzar a la puesta en marcha gradual que se realiza en semanas o meses. Algunos usuarios obtendrán ventajas del sistema nuevo antes que los otros: por ejemplo, un sistema médico que trata de llevar a cabo la interconexión de 10 o 15 clínicas diferentes en un hospital, pueden distribuir el proceso de cambio por etapas con un año de duración.

CONCLUSIONES

Muchos de los analistas, tenemos la costumbre de desarrollar sistemas sin ninguna planeación previa, sin realizar un análisis a fondo; si no por el contrario debido a la carga de trabajo o a las presiones que existen en los tiempos de entrega, vamos desarrollando malos hábitos de programación, y como consecuencia construimos sistemas con un ciclo de vida muy corto y conmuchos ajustes al momento de ponerlo en marcha.

La mayoría de las veces nos lleva más tiempo en mantenimiento los ajustes que el propio desarrollo, debido a la mala planeación y a un deficiente análisis y terminamos construyendo un sistema lleno de parches.

En este trabajo mostramos una de las metodologías para el desarrollo de sistemas que nos ayuda poder construir sistemas bien cimentados y con un mínimo de correcciones.

La metodología es como los planos arquitectónicos de una casa, donde antes de construir podemos ver como va a quedar, y así, poder realizar las modificaciones a tiempo y no tener que tirar muros y realizar las correcciones a lo ya constuido porque implicaría un mayor costo.

BIBLIOGRAFIA

PRESSMAN, S. ROGER, Ingeniería de Software, un enfoque práctico, ed. McGraw-Hill, primera edición, 1990.

FAIRLEY, E. RICHARD, Ingeniería de Sofware, ed. McGraw-Hill, 1987.

SOMMERVILLE, IAN., Ingeniería de Sofware, ed. Addison-Wesley Iberoamericana, segunda edición, 1988, México.

JOYANES AGUILAR LUIS, Fundamentos de Programación, ed. McGraw-Hill, primera edición, 1990 México.

FUNDACION ARTURO ROSENBLUETH, Programación Estructurada y diseño de Programas.

HERNANDEZ JIMENEZ RICARDO, Administración de Centros de Cómputo, ed. Trillas, tercera edición, 1991 México.

SENN A. JAMES, Sistemas de Información para la Administración, ed. Grupo Editorial Iberoamericana, tercera edición, 1990.

SENN A. JAMES, Análisis y diseño de sistemas de información, ed. McGraw-Hill, primera edición, 1990 México.

KENDALL E. KENNET Y KENDALL E. JULIE, Análisis y Diseño de Sistemas, ed. Prentice-Hall, primera edición 1991.

SQUIRE, ENID, Introducción al Diseño de Sistemas, ed. Fondo Educativo Interamericano, primera edición 1984 México.

BURCH, G. JOHN Y GRUDNITSKI, GARY, Diseño de Sistemas de Información, ed. Grupo Noriega Editores, primera edición, 1992 México.

ROBERT L. KRUSE, Data Structures and Program Design, ed. Prentice-Hall, tercera edición 1992.