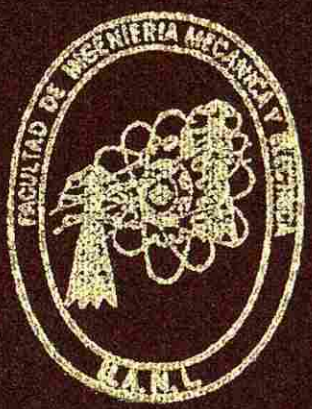
UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

ESCUELA DE GRADUADOS



INGENIERIA SOFTWARE

TESIS

QUE PARA OBTENER EL TITULO EN
LA MAESTRIA DE ADMINISTRACION CON
ESPECIALIDAD EN SISTEMAS

PRESENTA

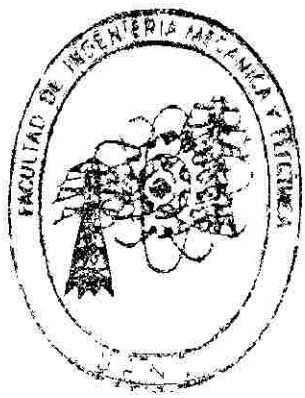
RAFAEL ANTONIO ROLDAN DECANINI

MONTERREY, N. L., MARZO DE 1986

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

ESCUELA DE GRADUADOS



INGENIERIA SOFTWARE

TESIS

QUE PARA OBTENER EL TITULO EN

LA MAESTRIA DE ADMINISTRACION CON

ESPECIALIDAD EN SISTEMAS

PRESENTA

RAFAEL ANTONIO ROLDAN DECANINI

MONTERREY, N. L. MARZO DE 1986

PARA MI FAMILIA :

DR. RAFAEL ROLDAN DE LA GARZA

SRA. ETNA DECANINI DE ROLDAN

ROSARIO

CARLOS

HUGO

ELIZA

SUZANA

ETNA

MARCELO

GUILLERMO

TERESA

TETI

CON GRAN CARIÑO.

A MIS TIOS Y PRIMOS

CON GRAN CARIÑO A MIS MAESTROS :

ING. JUAN ZAMORA

ING. VICTORIANO ALATORRE

ING. MARCO A. MENDEZ

ING. VICENTE GARCIA

I N D I C E

Pág

CAPITULO 1.0

INGENIERIA DE SISTEMAS COMPUTACIONALES. 1

1.1 CONSIDERACIONES SOFTWARE 3

1.2 APLICACIONES DEL SOFTWARE. 5

1.3 INGENIERIA SOFTWARE. 8

CAPITULO 2.0

PLANEAMIENTO DEL SISTEMA 13

2.1 FASE PLANEADORA 13

2.2 ANALISIS DEL SISTEMA 15

2.3 ESTUDIO DE FACTIBILIDAD. 16

2.4 ANALISIS COSTO BENEFICIO 18

2.5 ESPECIFICACION DEL SISTEMA 18

CAPITULO 3.0

PLANEACION DEL SISTEMA. 22

3.1 OBSERVACIONES SOBRE ESTIMACION . . . 22

3.2 EL OBJETIVO DEL SOFTWARE 23

3.3 RECURSOS 25

3.4 RECURSOS HUMANOS 26

3.5 HARDWARE 27

3.6 SOFTWARE 28

3.7 COSTEO SOFTWARE 30

I N D I C E

Pág

3.8 ACCESO DEL COSTEO 30

3.9 MODELOS DE ESTIMACION 32

3.10 MODELOS DE RECURSOS 33

3.11 MODELO DE ESTIMACION PUTMAN 35

3.12 MODELO DE ESTIMACION ESTERLING. . . . 37

3.13 MODELO DE COSTEO POR LINEAS DE CODIGO 40

3.14 TECNICA DE COSTEO POR ESFUERZO POR TAREA 42

3.15 COSTEO AUTOMATIZADO 43

3.16 PROGRAMACION. 46

3.17 METODOS DE PROGRAMACION 47

3.18 PLANEAMIENTO ORGANIZACIONAL 49

3.19 PLANEAMIENTO DEL SOFTWARE 51

CAPITULO 4.0

ANALISIS DE LOS REQUERIMIENTOS SOFTWARE.- 54

4.1 EL PASO DE ANALISIS DE REQUERIMIENTOS 54

4.2 TAREA DE ANALISIS 55

4.3 FLUJO DE INFORMACION. 57

4.4 DIAGRAMA DE FLUJO DE DATOS. 58

4.5 ESTRUCTURA DE INFORMACION 60

4.6 REPRESENTACIONES DE ESTRUCTURAS DE DATOS 60

4.7 DIAGRAMA DE BLOQUES JERARQUICOS . . . 60

I N D I C E

		Pág
4.8	DIAGRAMAS WARNIER.	61
4.9	REQUERIMIENTOS DE BASES DE DATOS . .	63
4.10	CARACTERISTICAS DE BASES DE DATOS. .	63
4.11	ESPECIFICACIONES DE REQUERIMIENTOS - DEL SOFTWARE	64
4.12	HERRAMIENTAS PARA EL ANALISIS DE REQUERIMIENTOS	66
4.13	SADT	67
4.14	HERRAMIENTAS AUTOMATIZADAS	68
4.15	SREM	68
4.16	PSL/PSA	69
CAPITULO 5		
	EL PROCESO DEL DISEÑO SOFTWARE . . .	72
5.1	LA FASE DE DESARROLLO.	72
5.2	EL PROCESO DE DISEÑO	74
5.3	DOCUMENTACION DEL DISEÑO	75
5.4	REVISION DEL DISEÑO	80
5.5	CONSIDERACIONES COSTO-BENEFICIO. . .	81
5.6	CRITERIOS DE REVISIONES DEL DISEÑO .	82
5.7	REVISIONES FORMALES.	84
5.8	REVISIONES INFORMALES.	86
5.9	INSPECCIONES	87
CAPITULO 6		
	CONCEPTOS SOFTWARE	89

I N D I C E

		Pág.
6.1	CUALIDADES DEL BUEN SOFTWARE. . . .	89
6.2	MODULARIDAD.	90
6.3	ABSTRACCION	91
6.4	INFORMACION ESCONDIDA	92
6.5	TIPOS DE MODULOS.	93
6.6	INDEPENDENCIA MODULAR	95
6.7	COHESION.	96
6.8	ACOPLAMIENTO.	98
6.9	MEDIDAS DEL SOFTWARE.	99
6.10	CIENCIA SOFTWARE DE HALSTEAD. . . .	100
6.11	MEDIDA DE COMPLEJIDAD DE McCABE . .	103
6.12	HEURISTICAS DEL DISEÑO.	105
CAPITULO 7		
	DISEÑO ORIENTADO AL FLUJO DE DATOS	109
7.1	DISEÑO Y FLUJO DE INFORMACION . . .	109
7.2	CONSIDERACIONES DEL PROCESO DE DISEÑO.	111
7.3	FLUJO DE TRANSFORMACION	111
7.4	FLUJO DE TRANSACCION.	112
7.5	ANALISIS DE TRANSFORMACION.	112
7.6	PASOS DEL DISEÑO.	112
7.7	ANALISIS DE TRANSACCION	114
7.8	PASOS DE DISEÑO	114
7.9	OPTIMIZACION DEL DISEÑO	115

I N D I C E

Pág.

CAPITULO 8

DISEÑO ORIENTADO A LA ESTRUCTURA DE DATOS, ... 118

8.1 EL DISEÑO Y LA ESTURCTURA DE DATOS 118

8.2 CONSIDERACIONES DEL PROCESO DE DISEÑO. 119

8.3 LA METODOLOGIA JACKSON. 120

8.4 CONSTRUCCION LOGICA DE PROGRAMAS. 124

8.5 DISEÑO DE DATOS 126

CAPITULO 9

HERRAMIENTAS DE DISEÑO DETALLADO. 130

9.1 CONSTRUCCIONES ESTRUCTURADAS. . . 131

9.2 HERRAMIENTAS GRAFICAS DE DISEÑO . 132

9.3 COMPARASION DE HERRAMIENTAS DE DISEÑO. 136

CAPITULO 10

LENGUAJES DE PROGRAMACION Y CODIFICACION. 139

10.1 EL PROCESO DE TRADUCCION. 139

10.2 CARACTERISTICAS DEL LENGUAJE DE - PROGRAMACION. 140

10.3 CLASES DE LENGUAJES 143

10.4 ESTILO DE CODIFICACION. 147

10.5 EFICIENCIA. 150

CAPITULO 11

PRUEBAS Y CONFIABILIDAD DEL SOFTWARE. 153

INDICE

Pág.

11.1 PRUEBAS DEL SOFTWARE. 154

11.2 PRUEBAS POR UNIDAD. 158

11.3 PRUEBAS POR INTEGRACION . . . 160

11.4 PRUEBAS DE VALIDACION 163

11.5 PRUEBAS DEL SISTEMA 164

11.6 EL ARTE DE DEBUGGING. 164

11.7 CONFIABILIDAD DEL SOFTWARE. . 166

CAPITULO 12

MANTENIMIENTO DEL SOFTWARE. 170

12.1 UNA DEFINICION DEL MANTENI - MIENTO SOFTWARE... 171

12.2 CARACTERISTICAS DEL MANTENI- MIENTO. 172

12.3 MANTENIBILIDAD 175

C A P I T U L O I

INGENIERIA DE SISTEMAS COMPUTACIONALES:

Ingeniería Software e Ingeniería Hardware son las actividades dentro de la categoría que llamaremos Ingeniería de Sistemas Computacionales. Cada una de éstas disciplinas representa intentar ordenar el desarrollo de sistemas basados en la computadora.

Técnicas Ingenieriles para el Hardware, desarrollado del diseño electrónico y han alcanzado un estado de madurez en un poco más de tres décadas. Técnicas del diseño del Hardware están bien establecidas, los métodos de fabricación son contínuamente mejorados, y una confiabilidad es una expectación real, en lugar, de una esperanza modesta.

En sistemas basados en computadora, el Software ha reemplazado al Hardware como el elemento del sistema más difícil de planear, menos probable a triunfar (en tiempo y dentro del costo), y muy peligroso para administrar. Y la demanda para el Software continúa inabatible; como crecen en número para sistemas basados en la computadora, complejidad y aplicación.

El objetivo del análisis y definición del sistema es descubrir el objeto del proyecto que tenemos. Esto se efectúa al no finar sistemáticamente la información a procesar, funciones requeridas, ejecución deseada, impedimentos del diseño y validación de los criterios.

Después que el objeto haya sido establecido, el-

Ingeniero en Sistemas Computacionales, debe de considerar un número de configuraciones alternativas que potencialmente pueden satisfacer el objetivo. Los siguientes criterios gobiernan la selección de la configuración de un sistema.

1. Consideraciones Comerciales ¿La configuración representa la solución con mayor ganancia? ¿Puede ser mercantilizado exitosamente?

2. Análisis Técnico, ¿Existe la tecnología para desarrollar todos los elementos del sistema? ¿Puede mantenerse adecuadamente la configuración? ¿Existen fuentes técnicas?

3. Evaluación de Fabricación. ¿Están disponibles las facilidades y equipo de fábrica? ¿La calidad puede efectuarse adecuadamente?

4. Problemas Humanos. ¿Hay personal capacitado disponible para el desarrollo y fabricación? ¿Existen problemas políticos? ¿El usuario entiende lo que el sistema va a desarrollar?

5. Interfaces Ambientales. ¿La comunicación entre máquina y máquina y humano-máquina son administrados en una forma inteligente?

6. Consideraciones Legales. ¿Pueden ser protegidos los aspectos de propiedad adecuadamente?

El peso de cada criterio anterior puede variar en cada sistema. Después de considerar los criterios, una configuración es seleccionada y las funciones sean asignadas en conjunto-

con los elementos potenciales del sistema.

CONSIDERACIONES SOFTWARE.

El Software es un elemento lógico, en vez de, físico. Por lo tanto, tiene características que son considerablemente diferentes que de las del Hardware:

* No hay una fase de fabricación significante para el Software; todos los costos están concentrados en el planeamiento y desarrollo.

* El Software no se "acaba"; hay pocas refacciones en el mundo Software.

* El mantenimiento Software a veces incluye modificaciones y beotifica los reportes.

Todos los lenguajes de programación son lengua - jes artificiales. Cada uno tiene un vocabulario limitado, una gramática explícitamente definido, un juego de reglas de sin - taxis y semántica bien formadas. Estos atributos son esencia - les para la traducción en la máquina. El lenguaje forma un componente de Software que se caracteriza como Lenguajes Máquina- y Lenguajes de Alto Nivel.

Lenguaje Máquina, es una representación simbólica del juego de instrucciones del CPU. Cuando un buen desarrollador de Software produce un programa, bien documentado y - - bien mantenible, el Lenguaje Máquina puede utilizar con extre-

ma eficiencia, la memoria y optimizar la velocidad de ejecu - ción del programa. Cuando el programa tiene un diseño pobre y tiene poca documentación el Lenguaje Máquina tiende a exacer - bar los problemas que puedan ocurrir.

Hasta el Lenguaje Máquina provee de velocidad - de ejecución y características de memoria, atractivas, tiene - un cierto número de desventajas serias:

1. El tiempo de implementación es dilatado
2. El programa resultante es difícil de leer
3. Pruebas difíciles
4. Mantenimiento extremadamente difícil
5. No es probable entre diferentes procesadores.

Existen más de 200 lenguajes de programación de Alto Nivel, pero menos de 10 lenguajes se utilizan ampliamente en la Industria. Estos lenguajes se pueden dividir en tres categorías:

Lenguaje de Fundación: Desarrollados entre 1950 y 1960, éstos lenguajes formaron una fundación para problemas científicos -- de propósito general y comercial. FORTRAN y COBOL son represen tativos de esta categoría. El ALGOL se puede considerar un -- lenguaje de fundación para la categoría de Lenguajes Estructu- rados.

Lenguajes Estructurados. Estos lenguajes emergieron por las fa llas en los Lenguajes Fundamentales, al reconocerlos y se de -

seaban buenas extenciones. Estructuras de datos sofisticados, - definición de subprogramas, estructuras de bloques y estructu - ras lógicas pueden acomodarse por éstos lenguajes. ADA, ALGOL,- PL/1, PL/M, PASCAL y C son representativos de esta categoría.

Lenguajes Especializados. Estos lenguajes proveen de características especiales la aplicación del Software ó son especiales por una virtud de una forma de Lenguaje Inusual ó Inconvencional. Los lenguajes que caracterizan esta categoría son: APL,- BLISS, LISP, RPG y SNOBOL.

El Assembler es el traductor para el Código Máquina, ejecutando relativamente simple tarea de convertir las- instrucciones simbólicas máquina a instrucciones-máquina ejecutables. Un Interper es un traductor que se utiliza para transformar Lenguajes de Alto Nivel, en la base de línea por línea. El APL y BASIC son uno de los Lenguajes que usualmente se ejecutan con un interpretador. El más común traductor del Lenguaje de Alto Nivel es el Compilador. Evaluando el programa globalmente, un Compilador, es capaz de optimizar el tamaño de memoria y/o la velocidad de ejecución de las instrucciones máquina que produce.

✓ APLICACIONES DEL SOFTWARE:

El Software puede ser aplicado en cualquier situación, en la cuál un juego de pasos (algoritmo) preespecificados han sido definidos. El contenido y la determinación de - la información son factores importantes para determinar la na-

turaleza de la aplicación del Software.

El término "Determinación de la Información" se refiere a la predicabilidad del orden y tiempo de la informa - ción. Un programa de Análisis Ingenieril acepta los datos, que tienen un orden predefinido, ejecuta los algoritmos analíticos sin interrupción y produce los datos resultantes en un formato tipo reporte ó gráfico. Tales aplicaciones son determinadas, - en un sistema operativo multi-usuario, acepta los datos, en -- los cuáles varía el contenido y el tiempo, ejecuta los algorit mos que pueden ser interrumpidos por condiciones exteriores y produce unos resultados que varían con la función del medio am biente y tiempo. Aplicaciones con éstas características son in determinadas.

Es algo difícil desarrollar categorías genéri - cas para las aplicaciones del Software. Las siguientes áreas - indican las aplicaciones potenciales.

Software del Sistema. Software del Sistema, es una colección- de programas escritos para auxiliar otros programas. Algunos - Softwares del Sistema procesan Estructuras de Información com- plejas, pero determinadas. Otras aplicaciones procesan gran -- cantidad de datos indeterminados. En cada caso el área del - - Software del Sistema, es caracterizado con muchas interaccio - nes con el Hardware de la computadora, uso pesado por usua -- rios múltiples, operaciones concurrentes que requieren planes, compartir las fuentes y un proceso sofisticado de administra - ción, estructuras de datos complejas e interfaces externas múl

tiples.

Software de Tiempo-Real. Software que mide, analiza y controla los eventos del mundo real como ocurren se llama Tiempo Real. - Los elementos del Software de Tiempo Real incluyen un componente de recolección de datos, que recolecten y formatea la información de un ambiente exterior, un componente de análisis que - transforma la información como la requera la aplicación, una -- componente de salida que responde al ambiente exterior y un componente de monitoreo que coordina todos los componentes para -- que la respuesta en tiempo real pueda mantenerse.

Software Administrativos. Procesamiento de información administrativa es el área más grande de aplicación. Sistemas discretos han evolucionado en Sistemas de Información Administrativos - - (MIS) que accesa una o más bases de datos grandes conteniendo - información administrativa.

Software Ingenieril y Científico. Software Ingenieril y Científico está caracterizado por algoritmos "crujientes de números". Las aplicaciones van desde Astronomía hasta Física Nuclear, de cálculos de tensión automotriz hasta Dinámica Orbital de naves espaciales, etc. Diseño auxiliado por computadora, simulación - de sistemas, y otras aplicaciones interactivas han iniciado tomar características de tiempo real.

Software Combinatorio. Software Combinatorio, hace uso de algoritmos no numéricos para resolver problemas complejos que - requieren de inteligencia artificial.

Reconocimiento de Patrones (Imágenes y Voces), Pruebas a Teoremas, Juegos y Prueba de Corrección de Software representan algunos de los problemas que son dirigidas por técnicas combinatorias.

INGENIERIA SOFTWARE :

Ingeniería Software, está moldeada en técnicas de Prueba de Tiempo, Métodos y Controles asociados con el desarrollo Hardware.

Aunque hay diferencias fundamentales entre Software y Hardware, los conceptos asociados con el planeamiento, desarrollo, revisión y control administrativo con similares para ambos elementos del sistema. Los objetivos principales de la Ingeniería Software son :

1. Una metodología bien definida que direcciona el ciclo de vida del Software, Planeación, Desarrollo y Mantenimiento.

2. Un juego establecido de componentes Software que documentan cada paso en el ciclo de vida y muestra larastreabilidad paso a paso.

3. Un juego de artificios predicables que pueden revisarse en intérvalos regulares a través del ciclo devida del Software.

En los siguientes parráfos, presentamos una meto

dología general de Ingeniería Software. Discutiremos brevemente cada paso en la metodología, la proyección del producto, y las revisiones que ocurren. Cada fase descrita corresponde a una fase del ciclo de vida del Software. El cielo de vida es un reconocimiento a largo plazo del Software, un reconocimiento que -- circunda las actividades que ocurren antes que inicie el desarrollo y después que el Software entre en uso.

Fase de Planeación. La Fase de Planeación, empieza con el paso de Planeación Software. Durante este paso una descripción limitada del esfuerzo del Software es desarrollada, los recursos requeridos para desarrollar el Software es predecida y el cálculo del costo y la programación son establecidas. El propósito del paso de planeación es de proveer una indicación preliminar de la viabilidad del proyecto en relación al costo y programación que ya han sido establecidos.

El siguiente paso en la Fase de Planeación, es - "Requerimientos de Análisis y Definición del Software". Durante este paso de elemento del sistema colocado al Software es defnido en detalle. El Flujo de Información y su estructura provee la llave a la definición de la interfase del sistema y las características funcionales del Software. Los requerimientos de ejecución o las limitaciones de los recursos son traducidos a las características del diseño Software. Análisis global del elemento Software, define los criterios de validación que se utilizarán - para demostrar que los requerimientos han sido cumplidos.

El análisis de los requerimientos del Software -

y su definición es un esfuerzo común entre el desarrollador y - el usuario. La especificación de los requerimientos del Softwa- re es la configuración producida por este paso.

La Fase de Planeación culmina con una revisión técnica de la especificación de los requerimientos del Softwa- re, conducidos por el desarrollador y el usuario. Una vez que la definición de los requerimientos sean establecidos, objeto, recursos y programación, identificados en el Planeación del -- Software son revaluados para su corrección. La información -- descubierta en el segundo paso puede impactar las estimaciones hechas en el primer paso.

Lo desarrollado en la Fase de Planeación sirve como fundación para la segunda fase en el proceso, desarrollo del Software.

Fase de Desarrollo. El primer paso de esta Fase, se concentra- en el Software, una estructura modular es desarrollada, las in terfaces son definidas y la estructura de datos es establecida. Un diseño heorístico es utilizado para evaluar cualitativamen- te el diseño. Este paso preeliminar de diseño es revisado pa ra obtenerlo completo y rastreabilidad a los requerimientos -- del Software. El documento de diseño es entregado y se con -- vierte en la configuración del Software.

Herramientas de Diseño son aplicadas para pro- veer una descripción detallada del diseño del elemento Softwa-

re . Cada descripción es adicionada al documento de diseño - después de su revisión.

Finalmente, después de dos pasos de desarro - llo, codificamos, esto es, generamos el programa con el uso- de un Lenguaje de Programación adecuado. La codificación es revisada para obtener claridad y estilo, pero debe de ser -- rastreable directamente a una descripción detallada del diseño. Un listado del lenguaje fuente para cada elemento modular del Software es entregado en el paso de codificación.

Los últimos tres pasos del desarrollo están - asociados con las pruebas del Software. Pruebas por unidad - intenta validar la ejecución de un componente modular individual del Software. Pruebas de Integración provee el asem -- blar de las estructuras modulares del Software, mientras - - prueba las funciones o interfaces. Prueba de Validación verifica que todos los requerimientos del Software, han cumplido. Una Prueba del Plan y Procedimiento se pueden desarrollar para cada paso de prueba. Una revisión de la documentación de prueba, casos de prueba y los resultados siempre son conducidos.

Fase de Mantenimiento. Esta Fase inicia antes de entregar el Software. Una revisión de la configuración del Software es -- conducida para asegurar que toda documentación está disponi - ble y adecuado para las tareas de mantenimiento que siguen. - La responsabilidad del mantenimiento es establecer y reportar

el plan por errores y define la modificación del sistema.

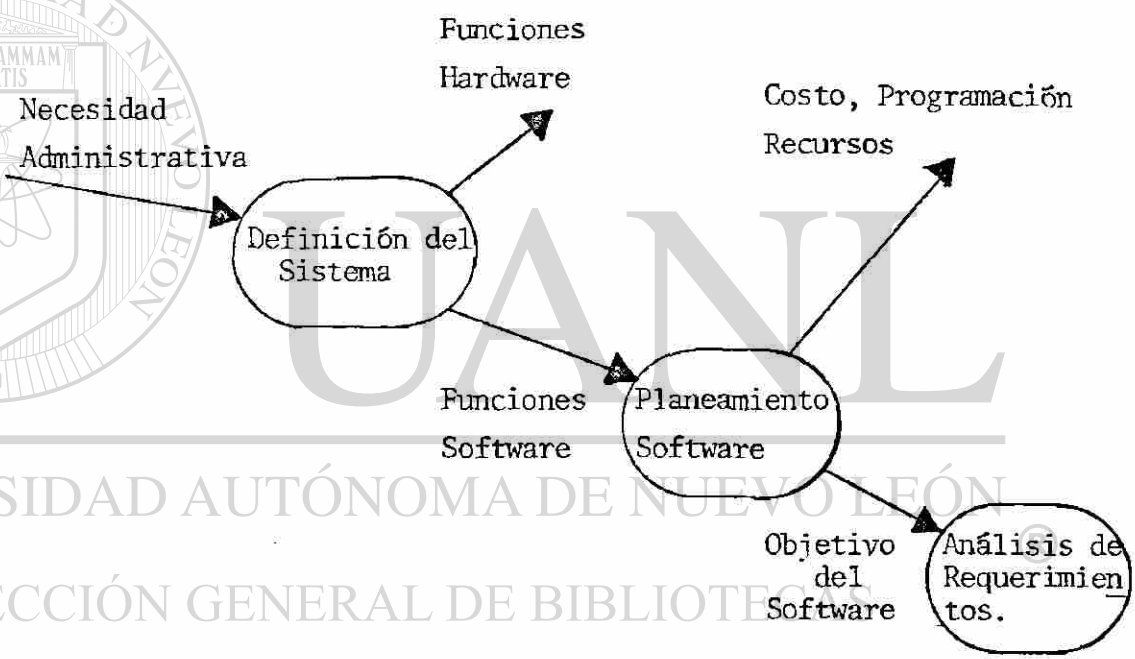
Las tareas asociadas con el mantenimiento Software dependen del tipo de mantenimiento a efectuarse. En todos los casos, la modificación del Software incluye toda la configuración (todos los documentos desarrollados en las fases de - planeación y desarrollo), no solamente el código.

C A P I T U L O II

PLANEAMIENTO DEL SISTEMA:

FASE PLANEADORA.

La Fase Planeadora del ciclo de vida del Software, es un proceso de definición, análisis, especificación, estima -- ción y revisión. El flujo de la Fase Planeadora se ilustra en la siguiente figura:



Especificación

Definición del sistema es el primer paso de la Fase - de Planeación en este punto el sistema se tomó como un conjunto. El paso de la definición del sistema procede a ambos, Ingeniería Soft - ware y Hardware. Los objetivos principales son :

1. Evaluar el concepto del sistema para Factibilidad, Costo-Beneficio, y Necesidad Administrativa.

2. Describir las interfaces, funciones y ejecución del sistema.

3. Ejecutar el análisis y diseño preeliminar del sistema.

4. Colocar las funciones al Hardware, Software y elementos suplementarios del sistema.

5. Establecer costos y límites de programación

Planeamiento Software y el análisis de requeri -- mientos del Software son potenciados por función y ejecución desarrollados durante el paso de definición del sistema. Pasos --- análogos ocurren en la definición del Hardware.

El objetivo primordial del planeamiento Software es el de estimar el costo y programación del desarrollo para el elemento Software del sistema. Para cumplir con este objetivo,- la tarea del Software debe ser completamente entendida y los recursos requeridos para satisfacerla deben de quedar bien definidos.

El último paso en la Fase Planeadora es el análisis de los requerimientos del Software. Una especificación detallada de los requerimientos del Software forma una fundación para la Fase de Desarrollo de la Ingeniería Software. Trabajando - en los límites de la tarea establecidos durante el planeamiento- Software, análisis de los requerimientos afina los detalles de - las interfaces Software, atributos funcionales, características- de ejecución, límites de diseño y validación de criterios.

Los tres pasos asociados con la fase de planeación, definición del sistema, planeación del Software, y especificación de los requerimientos del Software, requieren de una combinación de intuición, experiencia y tenacidad. Estas facciones, deben de acoplarse a un juego de tareas sistemáticas que permiten a la planeación que sea conducida en una manera controlada.

Definición del sistema no recae exclusivamente dentro de los pasos asociados con la Ingeniería Software. Las tareas de definición preveen un enfoque completo al sistema, colocando funciones a cada elemento del sistema.

La definición del sistema se caracterizan en tres tareas: Análisis, Colocación y Especificación. La primer tarea se enfoca en el entendimiento del problema y a la justificación de una solución propuesta. La segunda tarea evalúa las implementaciones alternativas de la solución basado en un criterio predefinido. La tercer tarea presenta la solución propuesta en la forma que sea revisable.

ANALISIS DEL SISTEMA:

El Análisis del Sistema comprende un número de tareas que definirá lo que se deba de cumplir, y si el cumplimiento es factible y cuál será el costo-beneficio del cumplimiento. Cada tarea requiere de análisis, seguido de evaluación y documentación.

El Análisis del Sistema es en sí, una actividad-

de solución de problemas que requiere de comunicación extensa - entre el desarrollador y el usuario. Esto se puede entender mejor, considerando los tópicos que son direccionados como se vaya analizando el problema y lo que se proyecta es el producto - del resultado de la solución del problema.

ESTUDIO DE FACTIBILIDAD :

Todos los proyectos son factibles, dándoles recursos ilimitados y tiempo infinito. Desafortunadamente, el desarrollo de un sistema computacional están plagados de faltas - de recursos y de fechas de entrega difíciles. Es necesario y -- prudente evaluar la factibilidad del proyecto lo más pronto posible. Meses ó años de esfuerzo, miles ó millones de dolares, y un desconcierto profesional se puede desviar si un sistema malconcebido es reconocido en la temprana etapa del planeamiento.

1 estudio de Factibilidad se concentra en cuatro áreas de interés :

Factibilidad Económica: Una evaluación del peso del costo de - desarrollo contra el beneficio derivado del sistema desarrollado.

Factibilidad Técnica: *Un estudio de función, ejecución y límites* que pueden afectar la habilidad de obtener un sistema aceptable.

Factibilidad Legal: Determinar si no existe una infracción, violación ó responsabilidad que pueda resultar del desarrollo del sistema.

Factibilidad Alternativas: Una evaluación de accesos alternativos para el desarrollo del sistema.

Un estudio de Factibilidad no está garantizado para sistemas en que la justificación económica es obvia, el riesgo técnico es bajo, algunos problemas legales son esperados, y no existe una alternativa razonable. Pero, si algunas de las condiciones anteriores fracasan, un estudio de esa área debe de conducirse.

La justificación económica es generalmente la última consideración para muchos sistemas. La justificación económica incluyen un amplio rango de problemas que incluyendo el análisis de costo-beneficio, estrategias de ingreso corporativo de largo plazo, el impacto en otros centros ó productos, recursos de costo necesarios para el desarrollo y el crecimiento del mercado potencial.

Factibilidad Técnica: Es frecuentemente la área más difícil de valorar en esta etapa del proceso de desarrollo del sistema. Porque los objetivos, funciones y ejecución son algo borrosas, todo es posible si las verdaderas asunciones son hechas. Es esencial que el proceso de análisis y definición sea conducido en paralelo con la factibilidad técnica. De esta forma, especificaciones concretas pueden ser calculadas como se vayan determinando.

Las consideraciones que son asociadas normalmente con la factibilidad técnica incluye :

Riesgo de Desarrollo. Se podrá diseñar el elemento - del sistema para que las funciones y ejecución necesarias sean- logradas dentro de los límites descubiertos durante el análisis.

Disponibilidad de Recursos. ¿Existe un Staff diestro, quién sea competente para desarrollar el elemento del sistema en question? ¿Existen otros recursos necesarios (Hardware y Software) disponibles para construir el sistema.

Tecnología. ¿Ha progresado la Tecnología al estado - - que soportaría el sistema?

Factibilidad Legal : Comprende un amplio rango de problemas que - incluyen contratos, condiciones, infracciones y millares de otras trampas que frecuentemente son desconocidos al Staff Técnico.

ANALISIS COSTO-BENEFICIO :

El Análisis de Costo-Beneficio, es complicado por criterios que varían con las características del sistema a desarro -- llar, el tamaño relativo del proyecto y la devolución esperada del capital que es deseado como parte plan estratégico. Además los mu chos beneficios derivados de los sistemas computacionales son in - tangibles. Comparaciones cuantitativas directas pueden ser difíci les de obtener.

ESPECIFICACION DEL SISTEMA :

La tercer tarea del paso de definición del sistema do

cumenta lo encontrado en los pasos anteriores. La especificación del sistema es lo primero que se puede entregar en el proceso. - Cada sección de importancia de la especificación se describe como sigue :

ESPECIFICACION DEL SISTEMA :

1.0 Introducción :

La sección introductoria describe los objeti - vos del sistema y el medio ambiente en que se espera que opere. - Esta sección también contiene un resúmen ejecutivo que especifica el objetivo del proceso de Desarrollo del Sistema, Factibilidad y Justificación, Recursos Requeridos y un Plan del Costo y Programación.

2.0 Descripción Funcional :

Una descripción de cada función del sistema es- descrita en esta sección. La descripción incluye una narración funcional que describe la información de entrada, tareas a ejecutar, información resultante, y datos de Interfase adicionales.

3.0 Colocación :

Cada función descrita en la sección 2.0 de la - especificación es colocada al elemento del sistema apropiado. Elementos de Hardware y Software son descritos separadamente. La in - formación, particularmente bases de datos o archivos existentes, - son descritas también.

4.0 Problemas :

Problemas Técnicos y Administrativos que afectan - el desarrollo del sistema son descritos en esta sección. Categorías típicas incluyen el ambiente exterior, interfaces, diseño e - implementación, recursos, y costo ó programación. Los problemas -- implican limitaciones. Esta sección debe de ser revisada cuidadosamente para asegurar que la implementación tenga éxito dentro de -- los límites especificados.

5.0 Costo :

Estimaciones del costo preciso pueden ser imposibles para determinar en esta etapa del proceso. El planeamiento - Software y su contraparte, Hardware, deberá de ser conducidos para indagar un costo estimado detallado.

Los límites de los costos son normalmente establecidos y anotados en esta sección.

6.0 Programación :

Una programación del desarrollo del sistema puede - ser predicado en una fecha de entrega determinada por demanda del usuario, impacto del mercado, ó fuerzas externas. Como el costo, -- una programación del desarrollo detallado no se puede establecer -- sin detallar el planeamiento del Hardware y Software. Información - cronológica conocida es definida en esta sección.

REVISION DE LA DEFINICION DEL SISTEMA :

La revisión de la definición del sistema evalúa la exactitud de la definición contenida en las especificaciones del sistema.

La revisión es conducida por el desarrollador y el usuario para asegurar que :

1. Enfoque del proyecto sea deliniado correctamente.
2. Funciones, ejecución, interfaces sean definidas propiamente.
3. El análisis del ambiente y el riesgo de desarrollo justifican el sistema.
4. El desarrollador y el usuario tengan las mismas perspectivas de los objetivos del sistema.

La revisión de la definición del sistema es conducido en dos segmentos. Inicialmente, es aplicado el punto de vista administrativo. Luego, una evaluación técnica de los elementos del sistema y funciones es conducido.

C A P I T U L O III

PLANEACION DEL SOFTWARE :

✓ Para conducir el proyecto de desarrollo del Software, debemos entender el objeto del trabajo que se va a desarrollar, los recursos requeridos, el esfuerzo y el costo que se requieran y la -- programación a seguir. La Planeación del Software, es el primero paso del proceso de Ingeniería Software, que provee al entendimiento.

✓ La Planeación del Software combina dos tareas: Investigación y Estimación. La Investigación nos permite definir el objeto del elemento Software del sistema. Utilizando las especificacio - nes del sistema como guía, cada función Software puede ser descrita.

Cuando hacemos estimaciones, estamos viendo el futuro y aceptamos algún grado de incertidumbre. Técnicas útiles para estimar el costo y la programación existen.

OBSERVACIONES SOBRE ESTIMACION :

Estimación de Recursos, Costo y Programación para el esfuerzo de desarrollo Software requiere experiencia, buena información, y el valor para tomar medidas cuantitativas cuando solo existe datos cualitativos.

La complejidad del proyecto tiene un fuerte efecto en la incertidumbre que es inherente en el planeamiento.

La complejidad, es una medida relativa que es afectada por esfuerzos pasados. Una aplicación de tiempo real puede ser perci

bida como exageradamente complicada a un grupo Software que se dedica a desarrollar aplicaciones Batch. La misma aplicación puede ser percibida como normal a un grupo Software dedicado a aplicaciones de tiempo real.

Tamaño del proyecto es otro factor importante que puede afectar la exactitud y la eficacia de las estimaciones. Como se incremente el tamaño, la interdependencia entre varios elementos del Software crecen rápidamente. El problema de descomposición, un acercamiento importante a la estimación, se convierte -- más difícil porque elementos descompuestos pueden ser formidables.

El grado de la estructura del proyecto también tiene un efecto en el riesgo de estimación. El riesgo es medido por el grado de incertidumbre en las estimaciones cuantitativas estableciendo Cursos, Costos y Programación. Si el objeto del proyecto es mal entendido a los requerimientos del proyecto están sujetos a cambios, la incertidumbre y el riesgo se convierte peligrosamente alta. El Planeador de Software, debe demandar que la función, ejecución y definiciones de interfase sean lo más completo posible, contenidas en las especificaciones del sistema. El Planeador como el usuario, deben de reconocer que la variabilidad en los requisitos, significa inestabilidad en los costos y programación.

EL OBJETIVO DEL SOFTWARE :

La primer tarea del Planeamiento Software, es la determinación del objetivo del Software. El Planeador debe des -

cribir el objetivo en un lenguaje que sea ambiguo y entendible - en los niveles Administrativos y Técnicos.

✓Una descripción del objetivo debe ser limitada,- esto es, los datos cuantitativos son establecidos explícitamente las limitaciones son anotadas y los factores mitigantes son descritos.

✓Las funciones del Software son evaluadas y en algunos casos redefinidas para mayor detalle. Porque las estimaciones de Costo y Programación son orientadas funcionalmente, y algún grado de descomposición es a veces usual. Consideraciones - de ejecución sitúan al constreñimiento del tiempo de proceso, limitaciones de memoria, y facciones dependientes especiales de la máquina.

✓La Función y Ejecución se deben evaluar juntas. La misma Función puede precipitarse en una diferencia en el es - fuerzo de desarrollo cuando es considerado en contexto de dife - rentes límites de ejecución.

El Planeador considera la naturaleza y complejidad de cada interfase para determinar cualquier efecto en el desarrollo de los Recursos, Costos y Programación.

El concepto de una interfase es interpretada - - como :

1. Hardware (procesador y periféricos) que ejecuta el Software y los aparatos (máquinas y displays) que indirectamente controla el Software.

2. Software que existe (rutinas que accesan la base de datos, paquetes de subrutinas, sistema operativo) y debe de acoplarse al nuevo Software.

3. Usuarios que utilizan el Software a través de terminales y otros equipos de entrada/salida.

4. Procedimientos que preceden al Software como una serie secuencial de operaciones.

En cada caso la información se transfiere a través de la interfase.

Si las especificaciones del sistema ha sido propiamente desarrollado, toda la información requerida para la descripción del objetivo del Software está disponible y documentado antes que el Planeador empieze. En los casos donde la especificación no ha sido desarrollada, el Planeador debe de tomar el rol del Analista de Sistemas para determinar los atributos y limitaciones que influirán en las tareas de estimación.

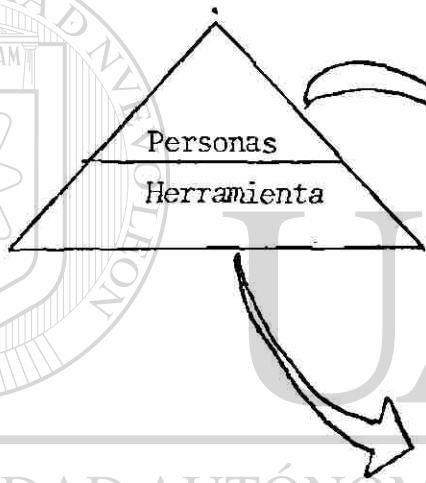
RECURSOS :

La segunda tarea de la planeación Software, es la estimación de los recursos requeridos para efectuar el esfuerzo de desarrollo del Software, la figura nos muestra los recursos de desarrollo como una pirámide. En la base, herramientas (Hardware y Software) deben existir para darle soporte al esfuerzo de desarrollo. En un nivel más alto, el recurso primario, personas, siempre son requeridas . Cada recurso es especificado con tres caracterís-

ticas :

1. Descripción del recurso
2. Tiempo cronológico que el recurso se requerirá
3. Duración de utilización del recurso

Las últimas dos características pueden ser vistas como la ventana del tiempo. La disponibilidad de los recursos - para una característica especificada debe ser establecida lo más temprano posible.



Recursos:

Especificar:

. Habilidad requerida

. Duración

. Fecha de inicio

Hardware

. Sistemas de desarrollo

. Requerimientos de utilización

Software

. Soporte

. Utilidad

RECURSOS HUMANOS :

El Planeador empieza evaluando el objetivo y seleccionando las habilidades requeridas para concluír el desarrollo.

Ambas posiciones organizacionales (Administrador, Gerente Soft - ware, etc.) y especializados (Telecomunicaciones, Base de Datos, Micro procesador, etc.) son especificados. Para proyectos relativamente pequeños (una persona-año ó menos) un individuo puede- ejecutar todos los pasos Software, consultando con especialistas como se requieran para proyectos grandes, la participación varía a través del ciclo de vida.

HARDWARE :

Tres categorías deben de considerarse durante la planeación del Software: Sistema de Desarrollo, Computadora Objeto, y otros elementos Hardware del nuevo sistema. El Sistema de Desarrollo es una computadora y sus periféricos que se utilizará durante la fase de desarrollo. El Sistema de Desarrollo es utilizado porque puede acomodar a usuarios múltiples, mantener gran - des volúmenes de información y contener un rico surtido de herramientas Software.

La Computadora Objeto es un procesador que ejecuta al Software como parte de un sistema basado en la computadora. En muchas aplicaciones la Computadora Objeto y el Sistema de Desarrollo son idénticas. Muchas aplicaciones de la microcomputadora requiere un sistema microprocesador de desarrollo (MDS) que - provee las facilidades para un Lenguaje de Alto Nivel, emulación dentro del circuito, y programación de PROM'S.

Otros elementos del sistema basados en la computadora pueden ser especificados como recargos para el desarrollo Software. Como ejemplo, el Software para un control numérico uti-

lizado en algunas clases de máquinas pueden requerir una máquina herramienta como parte de la prueba de validación, un proyecto - Software para litografías automatizadas pueden necesitar un foto litográfico en algún punto durante el desarrollo. Cada elemento Hardware debe de ser especificado por el Planeador.

SOFTWARE :

✓ La aplicación más temprana del Software en el desarrollo Software era el aprovechamiento de los elementos Software- del Sistema Operativo. El Traductor del Lenguaje Primitivo Assembly, fue escrito en lenguaje máquina y utilizado para desarrollar un assembler más sofisticado, ahora existen un gran surtido de herramientas Software; compiladores, editores, herramientas de prueba, apoyo de diseño y pre-ó post procesadores son solamente algunos de los recursos que pueden utilizar los desarrolladores del - Software.

✓ Dos categorías de Software que debe de considerar- el Planeador, la primera categoría, Software de apoyo, es utilizado para asistir en la fase de desarrollo. La segunda categoría, - Software de utilería, es adquirida de la librería y se convierte en parte del nuevo programa.

✓El Software de Apoyo, tiene un gran espectro de herramientas. La herramienta de apoyo más común es el compilador de lenguas de programación, éstos se han convertido en el pan de cada día para el desarrollador. El Software de apoyo existe para cada - paso del proceso de Software. Herramientas específicas de requeri-

mientos automatizados están disponibles para el paso de análisis, procesadores de lenguajes de diseño, generadores de diagramas de flujo, y simuladores son aplicados durante el diseño; Debuggers Dinamicos, Cross-Assemblers, Compiladores y Macro procesadores - son utilizados durante la codificación y la prueba de unidad; y vacíos analizadores se pueden utilizar durante las pruebas.

/Librerías del Software de utilería, existen para aplicaciones comerciales, sistemas y tiempo real y para problemas ingenieriles y científicos. Pero, hay pocas técnicas sistemáticas para adicionar a la librería, interfaces estándares para el Software de utilería son difíciles de enforzar, los problemas de calidad y de manutención se mantienen sin resolver, el desarrollador a veces desconoce el Software de utilería, que exista.

Dos reglas se deben considerar para el Planeador Software cuando el Software de utilería es especificado como recurso :

1. Si el existente Software de utilería está adentro- de los requerimientos, adquiéralo. El costo de adquisición del existente Software será casi siempre menos que el costo de desarrollar un Software equivalente.

2. Si el existente Software de utilería requiere de - algunas modificaciones, antes que pueda ser inge - grado al sistema, proceda con mucho cuidado. El -- costo de modificar el Software existente puede ser

a veces mayor que el costo de desarrollar un Software equivalente.

COSTEO SOFTWARE :

En los primeros años de la computación, los costos del Software comprometía un porcentaje bajo del costo total del sistema. La magnitud del error en la estimación del costo del Software tenía poco impacto. Ahora, el Software es el elemento más caro en el sistema. Grandes errores de estimación del costo hace la diferencia entre la ganancia y la pérdida. El sobregiro del costo puede ser desastroso para el desarrollador del Software.

El costeo del Software nunca será una ciencia exacta. Hay muchas variables, humanas, técnicas, ambientales y políticas, que pueden afectar el último costo del Software.

ACCESO DEL COSTEO :

El costeo es una tarea de estimación crucial en el Planeamiento del Software, el acceso al costeo debe procurar el más alto nivel de confiabilidad, o sea, el más bajo error en la estimación.

Para obtener estimaciones del costeo más confiables, un número de opciones potenciales podemos tomar :

1. Retrazar el costeo del Software, casi hasta finalizar el proyecto. (Obviamente, podemos obtener casi el 100% de la estimación después que se complete el proyecto).

2. Desarrollar un modelo paramétrico del costeo Software.

3. Utilizar técnicas de descomposición relativamente simples para generar el costo del proyecto en dos formas diferentes.

4. Adquirir un sistema automático de costeo.

Desafortunadamente, la primera opción, la más atractiva, no es práctica. Estimaciones del costo se deben dar al principio. La segunda opción, modelo de costeo ofrece un acceso potencial valioso de la estimación del costeo. Un modelo se basa en experiencia (datos históricos) y toma la forma:

$$c = f(v_i)$$

Donde c es el costo del Software y v_i son los parámetros independientes seleccionados que afectan al costeo. La tercera opción, técnicas de descomposición se accesan por el método de "dividir y vencer" de costeo. Descomponiendo el proyecto en funciones y tareas, el costeo (y programación) se puede ejecutar a pasos. El sistema automático de costeo provee una opción atractiva para estimar. En tales sistemas las características de la organización de desarrollo (experiencia) y el Software a desarrollar son descritos. Las estimaciones del costeo son derivadas de éstos datos.

Muchas características del Software pueden ser medidas, pero pocas nos dan un indicador adecuado de productividad macroscópico . La medida más simple y controversial de la productivi

dad es el número de líneas producidas por persona-mes. Esta medida solo se puede calcular después que el proyecto sea completado. Esto nos dá una indicación el esfuerzo humano gastado durante la - - planeación y desarrollo para producir una línea válida de código.

Muchos factores influencian a la productividad. Basili y Zelkowitz la definen en cinco categorías importantes :

Factor Personas. El tamaño y experiencia de la organización de desarrollo.

Factor Problemas. La complejidad del problema a resolver- y el número de cambios en los requerimientos del diseño.

Factor Proceso. Técnicas de Análisis y Diseño que se utilizan, lenguajes disponibles, y procedimientos de revisión.

Factor Productos. Confiabilidad y ejecución del sistema.

Factor Recursos. Disponibilidad de herramientas de desarrollo, recursos Hardware y Software.

MODELOS DE ESTIMACION :

Un modelo de estimación para el Software utiliza fórmulas derivadas empíricamente para predecir datos que serequieren como parte del paso de planeación. Desafortunadamente,- ningún modelo de estimación es apropiado para todas las clases -- de Software y en todos los ambientes de desarrollo. Los datos empíricos que apoyan a la mayoría de modelos son derivados de proyectos relativamente limitados. Por lo tanto, los modelos de estimación se deben de utilizar con sumo cuidado.

Modelos del proceso de desarrollo del Software pueden- ser derivados para predecir muchas características diferentes, - tales como la gente las requiere en función de tiempo, costo en- función de características del Software (medidas de complejidad, tamaño, y experiencia del Staff), duración del proyecto en fun- ción del tamaño de personas y ambiente de programación, esfuerzo en función de las características del Software, y calidad Soft - ware en función de las características del Software. Durante el paso de Planeación del Software, costo y programación son las -- características primarias del proyecto a ser estimadas. Entonces, los modelos que se relacionan en dollares ó esfuerzo (persona- - mes) para estimar los parámetros del Software son muy deseables.

MODELOS DE RECURSOS :

Los modelos de Recursos, consisten en una serie de ecua ciones derivadas para predecir al esfuerzo (persona-mes), dura -- ción del proyecto (en meses cronológicos), ó otros datos pertinen tes del proyecto. Basili describe cuatro clases de modelos de re- cursos: Modelos estáticos de una variable, modelos estáticos mul- tivariable; modelos dinámicos multivariables, y modelos teóricos.

El modelo estático de una variable toma la forma:

$$\text{Recurso} = c_1 * (\text{característica estimada})^{c2}$$

Donde el recurso puede ser esfuerzo E, duración de pro yecto D, tamaño del Staff S, ó líneas de Software documentadas -- DOC. las constantes c_1 y c_2 son derivadas de los datos recolecta- dos de proyectos pasados. Las características estimadas es líneas de código, esfuerzo ó otras características del Software.

Como ejemplo de un juego de modelos estáticos de una variable, consideramos el estudio de Walston y Félix. Basados en datos colectados de 60 proyectos de desarrollo con un rango de - tamaño de 4000 a 467000 líneas fuente y de 12 a 11758 personas-- mes, los siguientes modelos fueron derivados :

$$E = 5.2 * L^{0.91}$$

$$D = 4.1 * L^{0.36}$$

$$D = 2.47 * E^{0.35}$$

$$S = 0.54 * E^{0.6}$$

$$DOC = 49 * L^{1.01}$$

Donde, E esfuerzo (personas-mes), D duración del proyecto (meses calendario) y DOC en páginas de documentación, son - modelados en función de número de líneas fuente L. alternativamente, la duración del proyecto y los requerimientos del Staff (personas)S pueden ser calculadas del esfuerzo derivado ó estimado.

Las ecuaciones anteriores son específicas-ambientales y específicas-aplicaciones y no pueden ser aplicadas generalmente. Pero, modelos simples como los anteriores pueden ser derivados para un ambiente local si hay suficientes datos históricos disponi - bles.

Modelos estáticos multivariables, como su compañero anterior, hace uso de los datos históricos para derivar relaciones - empíricas. Un modelo típico de ésta categoría toma la forma:

$$\text{Recurso} = c_{1i} * e_1^{ci2} + c_{2i} * c_2^{ci3} + \ldots +$$

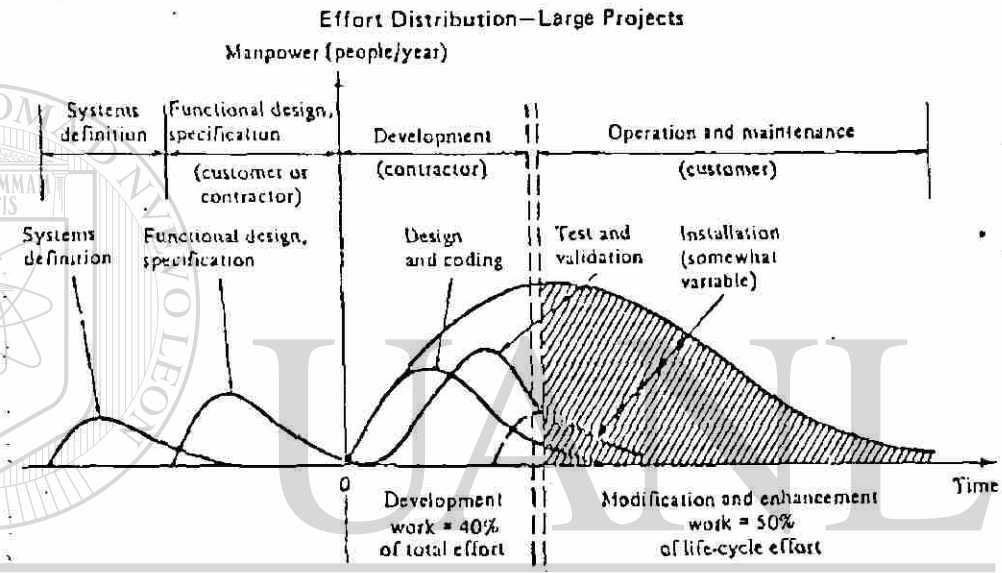
Donde ei es la "ïesima" característica de Software y c_{i1} y c_{i2} son constantes derivadas empíricamente para la ïesima característica.

El modelo dinámico multivariable proyecta los requeri -- mientos de los recursos en función del tiempo. Si el modelo es derivado empíricamente, los recursos son definidos en una serie de - pasos en tiempo coloca cierto porcentaje del esfuerzo (ó otro re - curso) a cada paso en el proceso. Cada paso puede ser subdividido- en tareas. Un acceso teórico al modelaje dinámico multivariable -- lo es hipotéticamente en la curva de gasto de recursos y de esta - curva se derivan las ecuaciones que modelan la conducta del recurso.

MODELO DE ESTIMACION PUTMAN :

El modelo de estimación Putman, es un modelo dinámico -- multivariable que asume una distribución específica del esfuerzo - sobre la vida del desarrollo del proyecto. El modelo se derivó de- distribuciones de trabajo encontradas en grandes proyectos (esfuerzo total de 30 personas-año o más). Pero, extrapolando a proyectos pequeños puede ser posible.

La distribución del esfuerzo para proyectos de Software grandes se pueden caracterizar en la siguiente figura:



(Source: L. Putnam, Software Cost Estimating and Life Cycle Control, IEEE Computer Society Press, 1980, p. 15. Reproduced with permission.)

La curva toma una forma clásica que describió analíticamente Lord Rayleigh. Datos empíricos en el desarrollo del sistema coleccionados por Norden, han sido utilizados para substanciar las curvas. Por lo tanto, la distribución del esfuerzo mostrado en la figura es generalmente llamada la curva de Rayleigh-Norden.

La curva de Rayleigh-Norden puede ser utilizada para derivar ecuaciones Software que relacionan el número de líneas código entregadas al esfuerzo y el tiempo de desarrollo :

$$L = C_k K^{1/3} T_d^{4/3}$$

Donde C_k es la constante del estado de tecnología y refleja los problemas que impiden al esfuerzo del programador. Valores típicos pueden ser: C_k = 6500 para un ambiente de desarrollo pobre (sin metodología, documentación y revisiones pobres o un modo de ejecución tipo Batch), C_k = 10,000 para un ambiente de desarrollo bueno (metodología buena, documentación y revisiones adecuadas y un modo de ejecución interactivo), y C_k = 12,500 para un ambiente excelente (herramientas,automatizadas, y técnicas). La constante L_k puede ser derivada para condiciones locales utilizando datos históricos coleccionados de esfuerzos de desarrollo pasados. Rearreglando la ecuación, llegamos a la expresión para el esfuerzo de desarrollo K:

$$K = L^3/C_k^3 * T_d^4$$

Donde T_d es el tiempo de desarrollo en años, y L es el número de líneas de código entregadas.

La ecuación para el esfuerzo de desarrollo se puede relacionar al costo de desarrollo, al incluír el costo de hombres-($/persona-año)

Por los elementos elevados a la tercera y cuarta potencia de este modelo, cualquier cambio pequeño en las líneas de -- código L pueden reducir significativamente el costo. Por ejemplo, si los requerimientos del costo se reducen, tal que, una reduc - ción del 10% en la estimación de líneas de código, el costo de - desarrollo global se reduciría en un 27%, de acuerdo al modelo - Putman. Se puede hacer el mismo cálculo si cambiamos la fecha de entrega del proyecto (cambia T_d) o si cambiamos la constante del estado de tecnología (cambio C_k).

MODELO DE ESTIMACION ESTERLING :

Esterling ha propuesto un modelo de productividad que- responde a las características microscópicas del ambiente de tra bajo. En un nivel personal el proceso de desarrollo es afectado- por "N" personas interactuando en el proyecto y las caracterísi cas del cambiente en el cuál éstas personas interactúan. Ester - ling afirma que juntas y otras actividades no productivas ocurren durante el período dc 8 horas de trabajo y que el mejor período - productivo ocurre en el tiempo extra.

Los parámetros asociados con el modelo Esterling in -- cluyen los siguientes elementos :

a. Fracción promedio del día de trabajo gastado en administración ó otros trabajos no directos con el proyecto.

t. Duración promedio de interrupciones de trabajo en minutos.

r. Tiempo promedio de recobro de interrupciones, en minutos.

k. Número de interrupciones en el dia de trabajo de personas involucradas directamente en el proyecto.

p. Número de interrupciones por jornada por otras causas.

i. Costo indirecto por persona expresado como fracción del pago base.

d. Pago diferencial de tiempo extra expresado en fracción del pago base.

La siguiente tabla reproducida de Esterling, nos muestra valores típicos para éstos parámetros :

Parámetros	Rango	Trabajadores	Programadores Optimísticos	Típicos	Pesimístic
a	0-0.5	0.0	0.05	0.10	0.15
t	1-20	3.0	3.0	5.0	10.0
r	5-10	0.5	0.5	2.0	8.0
k	1-10	1.0	2.0	3.0	4.0
p	1-10	1.0	1.0	4.0	10.0
i	1-3	0.2	0.2	0.5	1.0
d	1-2	1.5	1.0	1.0	1.5

El modelo de productividad consiste de cinco ecuaciones si g es el número promedio de horas extras trabajadas por día y N es el número de personas trabajando en el proyecto. Esterling desarrollo relaciones empíricas para la fracción del tiempo de trabajo utilizable W:

$$W = 1.25\ (8-8a+g-4r/60-P(T_{tr})\ /60 + K(N-1)\ (T_{tr})\ /60)$$

Utilizando W, las siguientes ecuaciones se desarrollaron :

$$c = NS\ (gd + 8C1+i)$$

$$e = NW/C$$

$$C = 1/e = c/NW$$

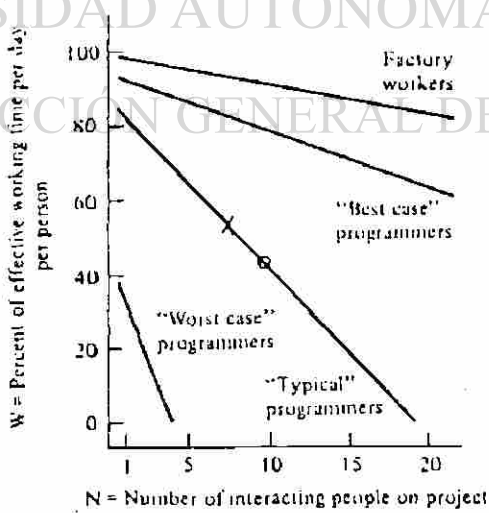
Donde : T = relación del tiempo calendario de personas-días para completar el proyecto

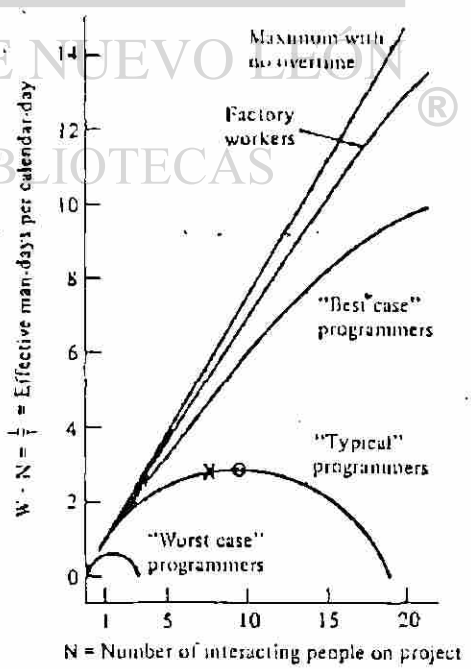
c = costo del trabajador por día para un salario -- promedio base

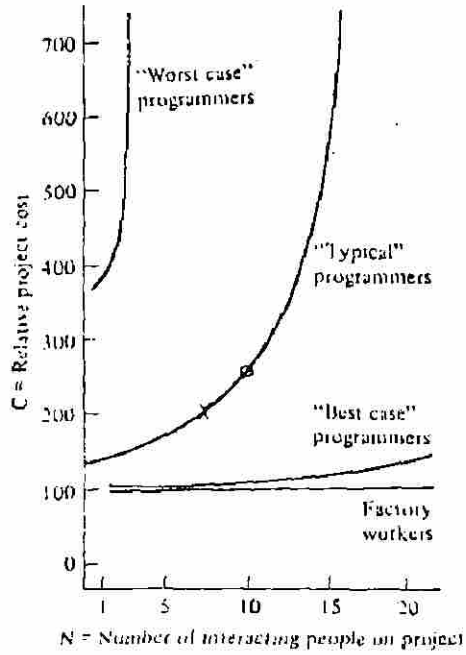
e = Eficiencia del costo

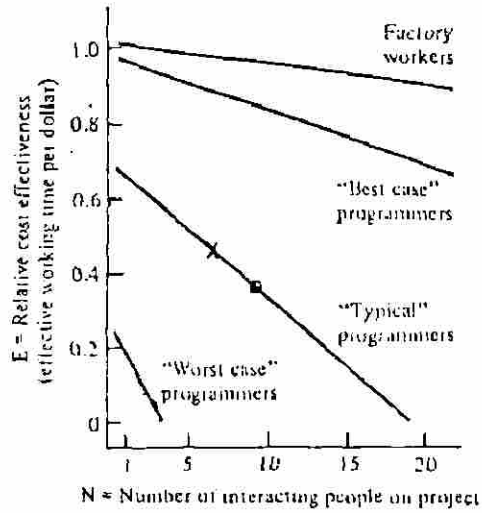
C = Costo del proyecto por persona-día

En las siguientes figuras nos ilustran el efecto del número de personas del proyecto con las variables descritas en - las ecuaciones. Las características del proyecto pueden seleccionarse para que el costo en tiempo del producto sea minimizado.









La naturaleza del tiempo de estudio de los parámetros - requeridos hacen difícil la colección de datos e impedirián la - aplicación del modelo Esterling. Además, el modelo no consideró las características del Software explícitamente. Pero, el modelo Esterling es el único acceso para la estimación del proyecto - - Software y provee una indicación útil de la eficiencia del ambiente local de programación.

TECNICAS DE COSTEO POR LINEAS DE CODIGO:

Cuando un Administrador de Desarrollo Software, se le pregunta que estime el costo de un proyecto nuevo, el debe tratar de ponerlo en dolares o alguna característica medible del Software. La técnica de costeo por líneas de código (LOC) liga el costo del dolar a la estimación del número de líneas de código fuente de programación que serán entregadas.

El objetivo del Software, determinado como parte del pago de planeación, provee una descripción de las funciones mayores. El primer paso en la técnica LOC especifica todas las funciones que serán implementadas en el código fuente. Las funciones -- serán descompuestas (subdivididas en elementos funcionales más pequeños) hasta que una estimación confiable del número de líneas - fuente requieran implementar la función.

La estimación de LOC provee un rango de valores para cada función descompuesta. Utilizando datos históricos ó intuición (cuando todo falla), el planeador estima optimísticamente y pesimísticamente el valor de LOC para cada función. Una indicación implícita del grado de incertidumbre es proporcionado cuando un rango de valores es especificado.

El número esperado (ó promedio) de LOC y la desviación del valor esperado son calculados en el siguiente paso de esta técnica de costeo. El número esperado de LOC L_e puede ser calculado como un promedio del optimístico a, del más probable m y del pesimístico b de las estimaciones del LOC:

$$Le = (a+4mtb)/16$$

Dando el mayor crédito a la estimación más probable y sigue una probabilidad de distribución Beta.

Asumiendo que hay poca probabilidad que el resultado actual del LOC caiga afuera de los valores optimísticos y pesimísticos. Por lo tanto, la desviación en las estimaciones del LOC L_2 (medida de la variancia en el resultado que se puede esperar) toma la forma estándar de :

$$Ld = \left(\sum_{i=1}^{N} \left(\frac{b-a}{6} \right)^2 \right)^{1/2}$$

Donde a y b son estimaciones para N funciones Software que se incluyen en la sumatoria.

Utilizando información histórica, el planeador selecciona el costo por LOC que caracteriza a cada función Software. Si tales datos no están disponibles, un valor promedio (de los recientes proyectos terminados) para que se pueda ser aplicado el $ 1/LOC. El promedio $/LOC puede ser corregida para reflejar los efectos inflacionarios, incrementar la complejidad del proyecto, nuevas personas ó otras características del desarrollo. En forma similar, el esfuerzo (expresado en personas-mes/LOC) pueden ser aplicadas a la estimación del LOC. El costo y esfuerzo pueden ser calculadas para cada función y la estimación total del costo y esfuerzo son determinados por el proyecto.

TECNICA DE COSTEO POR ESFUERZO POR TAREA :

El costeo por esfuerzo por tareas es la técnica más común para el costeo de cualquier proyecto de desarrollo ingenieril. Un número de personas-días -mes, ó -años es aplicado para la solución de cada tarea del proyecto. El costo en dolares es asociado a cada unidad de esfuerzo y el costo estimado se deriva.

Como la técnica de LOC, el costeo por esfuerzo por tareas empieza delineando las funciones del Software, análisis de -- requerimientos, diseños código y prueba, deben de ejecutarse para cada función. Las funciones y tareas relacionadas se pueden representar en la siguiente tabla:

Tareas Funciones	Análisis de Requerimientos	Diseño	Código	Pruebas	Totales
Total					*
Relación ($)					
Costo					**

* Esfuerzo estimado para todas las tareas.

** Costo estimado para todas las tareas.

El segundo paso del esfuerzo por tareas establece el esfuerzo (personas-mes) que se requirirán para completar cada tarea. Estos datos serán la parte central ó matriz central de la tabla anterior.

El tercer paso de esta técnica enlaza los costos por unidad de esfuerzo a cada tarea del proyecto. Es muy probable que éstos costos varían con cada tarea.

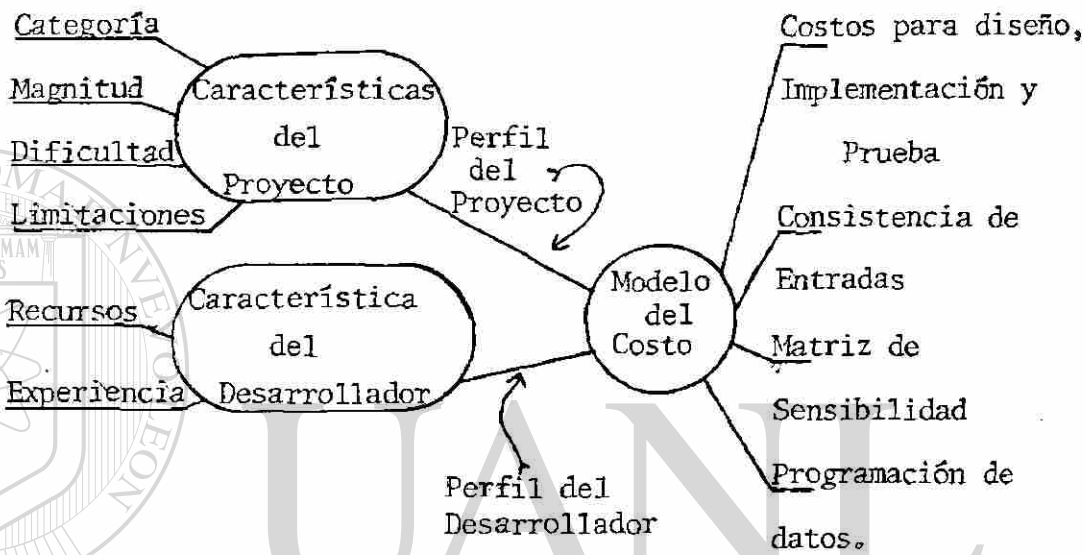
En el último paso son calculados para cada función y tarea el costo y esfuerzo. Si el costeo por esfuerzo por tarea es ejecutado independientemente al costeo por LOC, tendremos dos estimaciones del costo y esfuerzo que se pueden comparar y reconciliar. Si ambas técnicas nos muestran una concordancia considerable, hay razón suficiente que las estimaciones son confiables. Pero, si en los resultados hay mucha variación, hay que seguir investigando y analizando.

COSTEO AUTOMATIZADO :

Un número creciente de organizaciones Software han de

sarrollado o adquirido un sistema de costeo automatizado. La configuración de un sistema de costeo típico se ilustra en la siguiente figura.

El modelo del costo acepta la información del proyecto y del desarrollador y produce una estimación del costo del Software.



Inicialmente, las características de la organización de desarrollo son descritas en una forma cuantitativamente. Recursos de desarrollo, experiencia del Staff de Software, número de responsabilidades concurrentes, estructura organizacional, y otras características son utilizadas para producir el perfil del desarrollador. Para calibrar con mayor presición el ambiente local de desarrollo, el modelo de costeo puede ser ejecutado en reversa, utilizando datos coleccionados de un proyecto terminado. Tales datos pueden ser utilizados para afinar los coeficientes del modelo para reflejar mejor a la organización local. Después que se haya establecido el perfil del desarrollador, las características del proyecto se especifican. Area de aplicación del Soft -

ware, magnitud y complejidad del proyecto, limitaciones de diseño y programación, características de ejecución, limitaciones -- del Hardware y otros datos que componen al perfil del proyecto.

Los perfiles del desarrollador y del proyecto proveen las entradas primarias para el modelo de costeo automatizado. El modelo provee los costos para el diseño del Software, - implementación, y prueba. Además, de la información del costo -- básico, el modelo también provee la programación del proyecto, - una serie de reportes de varianza que indican un riesgo estimado y la evaluación de la consistencia de entrada. Si los datos da - dos para establecer los perfiles del desarrollador y del proyecto son precisos, el modelo automatizado produce resultados - - excelentes.

SLIM es un sistema de costeo automatizado -- que rebasa en la curva de RayLeigh-Norden para el ciclo de vida del Software y el modelo de estimación de Putman. SLIM aplica- las técnicas del modelo de Putman; programación lineal, simulación estadística y el método de programación PERT para derivar- las estimaciones del proyecto Software. El sistema habilita al- planeador del Software a ejecutar las siguientes funciones en - una terminal interactiva :

1. Calibrar el ambiente local de desarrollo, interpretando los datos históricos dados por el planeador.
2. Crear un modelo de información del Software a desarrollar, obtenidos de las características básicas de Software, atributos personales, y consideraciones ambientales.

3. Colar la conducta del Software, el método utilizado por SLIM es más sofisticado, automatizado, que la versión de la técnica de costeo de LOC.

Una vez que el tamaño del Software ha sido establecida, SLIM computa la desviación del tamaño, una indicación de la estimación de la desconfianza, el perfil de sensitividad que indica una desviación potencial del costo y esfuerzo y un chequeo de consistencia con los datos coleccionados para sistemas de Software de similar tamaño.

El Planeador puede invocar el análisis de programación lineal que considera los límites de desarrollo en el costo y esfuerzo. Utilizando la curva de Rayleigh-Norden como modelo, SLIM también nos provee con distribución el de mes a mes del esfuerzo y costo para que los requerimientos del Staff y el flujo de efectivo puedan ser proyectadas.

PROGRAMACION :

La Programación para el proyecto de desarrollo del Software se pueden observar de dos diferentes perspectivas. En la primera una fecha final para liberar el sistema que ha sido establecido. La organización Software está limitada a distribuír el esfuerzo dentro del marco del tiempo prescrito. La segunda perspectiva la programación del Software asume un ligamiento cronológico que se había discutido pero la fecha de liberación es definida después de análisis cuidadoso del elemento Software.

METODOS DE PROGRAMACION :

Programación del proyecto Software no difiere- mucho de cualquier esfuerzo de desarrollo multitareas. Entonces, las herramientas generalizadas de programación y técnicos pue - den ser aplicadas al Software con poca modificación.

La evaluación del programa y técnica de revi -- sión (PERT) y el método del camino crítico (CPM) son dos métodos de programación del proyecto que pueden ser aplicados al desarro llo del Software. Ambas técnicas desarrollan una descripción en cadena del proyecto, esto es, una representación pictorial ó tabular de las tareas que deben acompletarse del inicio al final - del proyecto. La cadena es definida desarrollando una lista de- todas las tareas del proyecto específico y una lista de ordenes (a veces llamada lista de restricciones) que indica en que orden se deban acompletar cada tarea.

El PERT y CPM, ambos, proveen de herramientas- cuantitativas que permiten al planeador a:

1. Determinar el camino crítico, una cadena de tareas que -- determinan la duración del proyecto.
2. Establecer las estimaciones más probables de tiempo para tareas individuales, aplicando los modelos estadísticos.
3. Calcular los tiempos límites que definen la ventana del - tiempo para una tarea en particular.

Cálculos del límite de tiempo pueden ser muy uti

lizables en la programación del proyecto. Un desliz en el diseño de una función, por ejemplo, puedo retardar el desarrollo de - - otras funciones. Riggs describe los límites de tiempo que se pue den discernir de las cadenas PERT o CPM:

1. El tiempo más temprano que una tarea pueda empezar cuando todas las tareas anteriores sean completadas en el menor tiempo posible.

2. El tiempo más tardado para la iniciación de la tarea antes que el tiempo mínimo de conclusión del proyecto.

3. La conclusión más temprana, suma de los juicios más tempranos y la duración del proyecto.

4. La conclusión más tarde, los inicios más tardes adicionados a la duración de la tarea.

5. La flotación total, el tiempo de sobra ó el abatimiento - permitido en la programación de tareas para asegurar que el camino crítico de la cadena es mantenida dentro de la programación.

Cálculos del límite de tiempo nos dejan de - terminar el camino crítico y provee al administrador de un método cuantitativo para evaluar el progreso, como se vayan completando las tareas.

Como un comentario final de programación, recordamos la curva de Rayleigh-Nordon, representa el esfuerzo gas--

tado durante el ciclo de vida del Software. El planeador debe - - reconocer el esfuerzo gastado en el Software no termina al fin - del desarrollo. El esfuerzo de mantenimiento, no es fácil de programar en este estado, será el de mayor costo sobre el ciclo de vida del Software. La meta primaria del Ingeniero en Software, - ayudará a reducir este costo.

PLANEAMIENTO ORGANIZACIONAL :

Existen muchas estructuras organizacionales humanas para el desarrollo del Software como hay organizaciones -- que desarrollan el Software. Para mejor ó peor, las estructuras no pueden ser modificadas fácilmente. La incumbencia son consecuencias prácticas y políticas de la organización, los cambios - no entran en las responsabilidades del objetivo del planeador. Pero, la organización de las personas involucradas directamente con un proyecto nuevo puedan ser consideradas durante el paso de planeación.

Las siguientes opciones están disponibles para la aplicación de recursos humanos a un proyecto que requerirá N personas trabajando K años:

1. N individuos son asignados a m tareas funcionales diferentes, relativamente poco trabajo ocurre, y la coordinación es responsabilidad del administrador del -- Software.

2. N individuos son asignados a m tareas funcionales diferentes (m $\leq$ N), asi que, equipos informales son establecidos, y la coordinación entre los equipos son la res -

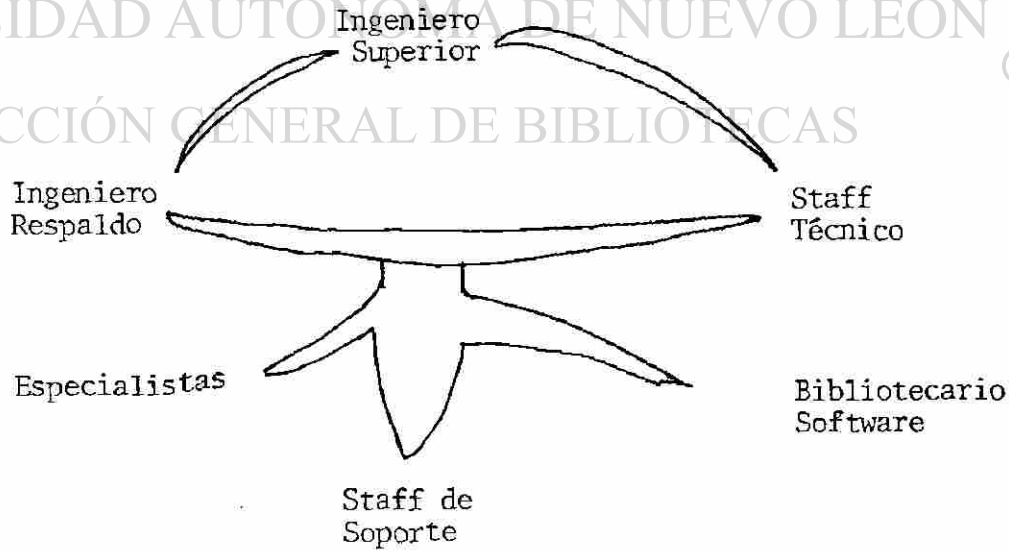
ponsabilidad del administrador del Software.

3. N individuos organizados en K equipos, cada equipo se - le asigna una ó más tareas funcionales y tiene una organización específica, la coordinación es controlada por- ambos, el equipo y el administrador del Software.

Aunque es posible decir argumentos en pro y -- con para cada uno de los accesos, hay una creciente evidencia que indica que equipos formales de la organización (opción 3) son los más productivos.

El equipo de desarrollo Software tiene sus orígenes en el equipo de programación superior, concepto primero propuesto por Harlan Mills y descrito por Baker. El equipo de desarrollo se ilustra en la siguiente figura.

Equipo de Desarrollo Software
(Equipo de Programación Superior)



El núcleo del equipo es el Ingeniero Superior (Jefe - Programador) quién planea, coordina y revisa todas las actividades técnicas del equipo, Staff Técnico (normalmente de dos a - cinco personas) quién conducen las actividades de análisis y desarrollo, y el Ingeniero de Respaldo quién soporta las actividades del Ingeniero Superior y puede reemplazarlo con una pérdida mínima de continuidad del proyecto.

El equipo de desarrollo puede pedir el servicio de -- uno o más especialistas (expertos en telecomunicaciones ó diseñadores de bases de datos), Staff de soporte y el bibliotecario -- Software.

El bibliotecario sirve a muchos equipos y ejecuta las siguientes funciones: Mantiene y controla todos elementos de la- configuración Software, tal como, la documentación, listados - - fuente, datos, y medios magnéticos, ayuda a coleccionar y formatear los datos de productividad Software, cataloga o indexa módulos revisables de Software, y asiste a los equipos en la investigación, evaluación y en la preparación de documentación. La importancia del bibliotecario no puede ser sobre enfatizada. Este- actúa como controlador, coordinador y potencialmente, en el evaluador de la configuración Software.

PLANEACION DEL SOFTWARE :

Cada paso en el proceso del Software debe producir el Software que pueda ser revisado y actúan como cimentación para - los pasos siguientes. El Planeamiento Software es producido en-

la culminación del paso de planeación. Proyee de la información del costo básico y de programación que serán utilizados a través del ciclo de vida del Software.

La Planeación del Software es un documento -- que se direcciona a una audiencia diversa. Esto debe hacer :

1. Comunicar el objetivo y los recursos a la administración,- del Staff Técnico y al usuario.

2. Define el costo y programación para la revisión de la administración.

3. Provee un acceso general del desarrollo Software para todas las personas asociadas con el proyecto.

Una representación del costo y programación - variará con la audiencia a quién se dirige. Si el plan es utilizado solo como un documento interno, los resultados de cada - - técnica de costeo puede ser presentada. Cuando el plan es desanimado afuera de la organización, se provee con una síntesis de los resultados de las técnicas de costeo. Similarmente, el grado de detalle contenida en la sección de programación puede variar- con la audiencia y formalidad del plan.

El plan del Software no necesariamente tiene - que ser un documento largo y complejo. El propósito es ayudar -- a establecer la viabilidad del esfuerzo de desarrollo.

El plan del Software se concentra en el estatuto general de "que" y en los estatutos específicos de "cuanto" -

y "cuanto tiempo" . Pasos subsecuentes en el proceso se concentrarán en "como" .

C A P I T U L O 4

ANALISIS DE LOS REQUERIMIENTOS SOFTWARE

Las especificaciones completas de los requerimientos del software son esenciales para el éxito del esfuerzo de desarrollo del software. No importa lo bien diseñado, o lo bien codificado, un programa pobremente especificado desalienta al usuario y pena al desarrollador.

La tarea de análisis de los requerimientos es un proceso de descubrir y evaluación. El objetivo software, es inicialmente establecido durante la planeación del software, y es refinado en detalle. Soluciones alternativas son analizadas y colocadas en varios elementos software.

Ambos, el desarrollador y el usuario, toman un rol en -- las especificaiones del software. El usuario trata de reformular, a veces , un concepto nebuloso de la función software y ejecución en detalle concreto. El desarrollador actua como un interrogador, consultante, y resolvedor de problemas.

PASO DE ANALISIS DE REQUERIMIENTOS

El análisisde los requerimientos es el último paso de la fase de planeación del ciclo de vida del software. Utilizando el objetivo del software como guía, el análisis de los requerimientos del software trata de satisfacer los siguientes objetivos:

- Proveer una fundación para el desarrollo del software descubriendo el flujo y la estructura de la información.

- Describir el software identificando los detalles de interfase, proviendo una descripción a fondo de las funciones; determinando los límites del diseño y definir la validación de los requisitos del software.

- Establecer y mantener la comunicación entre el usuario y el desarrollador para que los dos objetivos anteriores se satisfagan.

Para completar estos objetivos, el paso de análisis de los requerimientos para por una serie de tareas que discutiremos en seguida:

TAREAS DE ANALISIS

El análisis de los requerimientos del software puede ser dividido en cuatro áreas de esfuerzo:

1.- Reconocimiento del problema.

2.- Evaluación y síntesis.

3.- Especificación.

4.- Revisión.

Inicialmente, el analista estudia las especificaciones del sistema (si alguna existe) y el plan del software. Es importante entender el software en el contexto del sistema y revisar el ob-

jetivo del software que fue utilizado para generar las estimaciones de laplaneación. Luego, comunicación para elanálisis se debe esta-- blecer para asegurar el reconocimiento del problema.

El analista debe establecer contacto con la administra- - ción y el staff técnico de la organización del usuario y la organización de desarrollo. El administrador del proyecto puede seguir como coordinador para facilitar los caminos de comunicación. La meta del analista es el reconocimiento de los elementos básicos del problema, como los percibe el usuario.

La evaluación del problema y la síntesis de la solución - es la siguiente area de esfuerzo para el análisis. El analista debe evaluar el flujo y la estructura de la información, refinir todas - las funciones software en detalle, establecer las características - de las interfases del sistema y de descubrir los límites del diseño. Cada una de éstas tareas sirven para describir el problema para sintetizar la solución.

Las tareas asociadas con la especificación se esforzan en la representación del software que pueda ser revisado y aprobado -- por el usuario. En el mundo ideal, el usuario desarrolla las especificaciones de los requerimientos del software completamente. Este - es un raro caso en el mundo real. Las especificaciones son desarrolladas en un esfuerzo conjunto entre el desarrollador y el usuario.

Cuando son descritas, las funciones básicas, ejecución. - interfases e información , criterios de validación son especifica-- dos para demostrar un entendimiento de la implementación del soft-

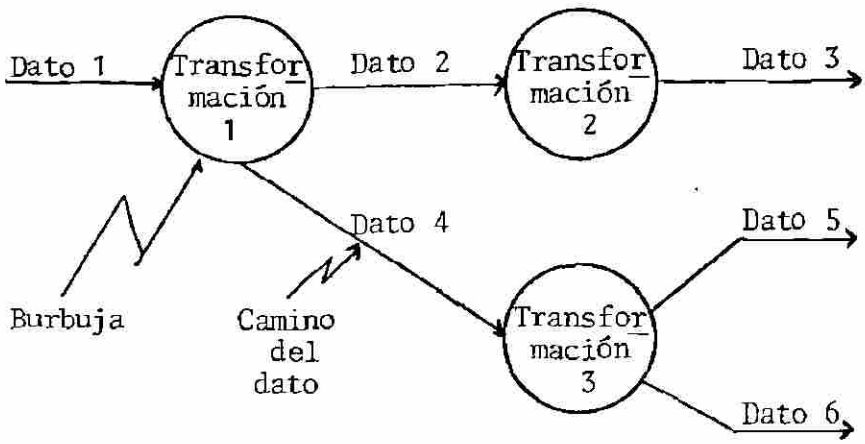
ware exitosa. Estos criterios sirven como base para pruebas durante el desarrollo software. Son escritos formalmente las requerimientos de la especificación para definir las características y atributos - del software. Además se escribe un manual del usuario preliminar.

Se puede parecer raro que el manual del usuario se desarro lle tan temprano en el proceso del software. Esto para el analista (desarrollador) vea el punto de vista del usuario del software,- particularmente sistemas interactivos importantes.

Una vez que lasespecificaiones y el manual del usuario han sido desarrollados, la revisión de las especificaciones casi siempre resulta con modificaciones . El impacto de la información de los requerimientos en las estimaciones del costo y programación (Plan - del Software) es también revizado. Cuando sea necesario, las estimaciones son modificadas para reflejar el nuevo conocimiento.

FLUJO DE INFORMACION

Como se mueva lainformación a travéz del software, es modificado por una serie de transformaciones. El diagrama de flujo de - datos es una técnica gráfica que representa el flujo de información y las transformaciones que se aplican como los datos se mueven de - la entrada a la salida. La forma básica del diagrama de flujo de datos se ilustra en la siguiente figura .



El diagrama es similar en forma a otros diagramas de flujo de actividades y ha sido incorporado en las técnicas de análisis y si seño propuestos por Yourdon, Constantine y DeMarco. También es conoci do como gráfica de flujo de datos o carta burbuja .

Un modelo fundamental del sistema puede ser representado - por el diagrama de flujo de datos. El elemento entero del software se representa como una burbuja con flechas de entrada y de salida. En ge neral, el modelo fundamental es refinido en una serie de burbujas, re presentando cada una, una transformación que ocurre a través de los - caminos del flujo de información de entrada a la salida.

DIAGRAMA DE FLUJO DE DATOS

El diagrama de flujo de datos tiene tres atributos:

- Flujo de información en cualquier sistema, manual, automático o hi brido, puede ser representado.

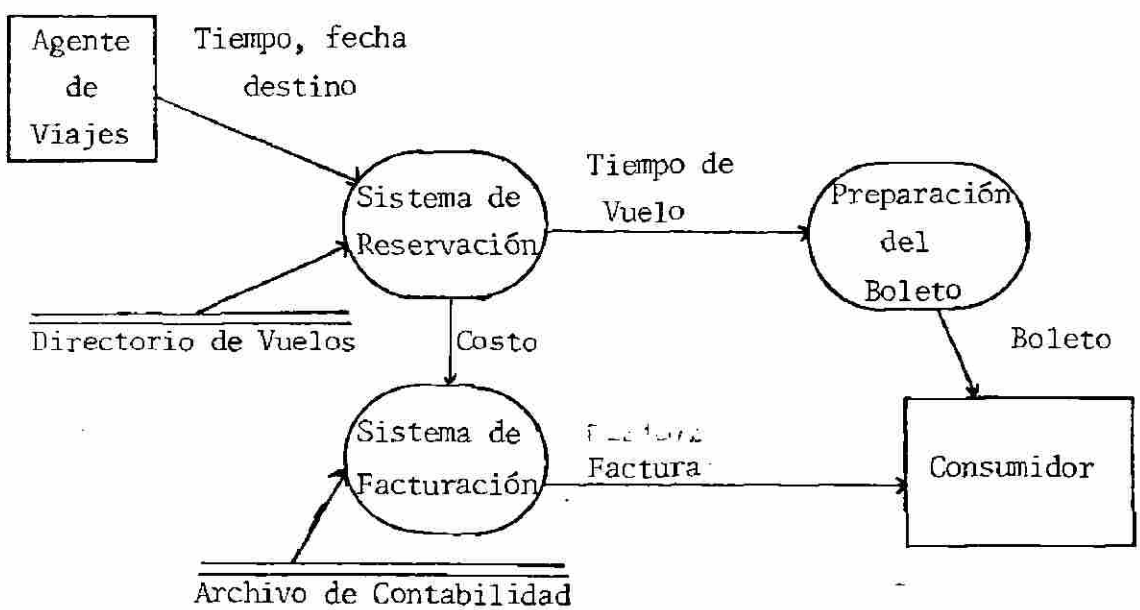
- Cada burbuja puede requerir algo de refinamiento para establecer --

una compresión total.

. El flujo de datos es enfatizado, en lugar de un flujo de control.

Las herramientas gráficas para representar el diagrama de flujo de datos son bastante simples. La información (flujo de datos) se representan por flechas marcadas. El proceso (transformación) se representan por burbujas rotuladas. Fuentes y depósitos de información se distinguen por cajas marcadas e información guardada (archivo de datos) se representa por una doble subraya. La fuente de información es donde se originana los datos , y el depósito de información es donde termina la información.

Un diagrama de flujo de datos utilizando las características anteriores se muestra en la siguiente figura. En el ejemplo, el agente de viajes (fuente de información) provee los datos al sistema de viajes representado por tres transformaciones (burbujas). Los datos del viaje (tiempo, fecha y destinación) son transformadas a un boleto que fluye al consumidor (un depósito de información). Los datos guardados (directorio de vuelo y el archivo de contabilidad) proveen otros caminos de información.



ESTRUCTURA DE INFORMACION

La estructura de información es una representación de una relación lógica entre los elementos individuales de los datos. Porque la estructura de la información afectará invariablemente al diseño final del software, una consideración en la estructura de la información es esencial para el éxito del análisis de requerimientos del software.

La estructura de datos dicta la organización, métodos de acceso, grado de asociatividad y procesos alternativos para la información.

REPRESENTACIONES DE ESTRUCTURAS DE DATOS

Durante el análisis de los requerimientos del software, una estructura jerárquica de información es encontrado a veces. Porque ésta estructura de información puede tener un impacto significante en los requerimientos del diseño del software, el analista debe representar la jerarquía en un modo ambiguo y legible. Hay dos métodos de representación de estructuras jerárquicas de datos, el diagrama de bloques jerárquico y el diagrama Warnier. Es interesante notar que estas formas de diagrama se puede utilizar para representar los datos y el software.

DIAGRAMA DE BLOQUES JERARQUICOS

Este diagrama representa la información como una serie de bloques organizados en multinivel, como una estructura, un bloque es utilizado para representar toda la jerarquía. Los niveles siguientes contienen los bloques que representan varias categorías de información que pueden ser vistas como un subjuego de bloques en el árbol. En el nivel

más bajo del diagrama, cada bloque contiene datos individuales.

El diagrama de bloques jerárquico se representa en más detalle al refinarse la estructura. Este modo de representación es mejor para el análisis de requerimientos. El analista empieza catalogando la información del nivel superior. La refinación contínua en cada camino del diagrama hasta que toda la información detallada sea establecida.

Pero, los medios técnicos de los requisitos sean satisfechas no se indican. El diagrama de bloques provee de poca información de las características físicas de la estructura de datos. Indicadores de archivos y de registros, formado de datos y tributos internos no son especificados. Un modelo detallado de la estructura de datos es desarrollada normalmente como parte del diseño del software.

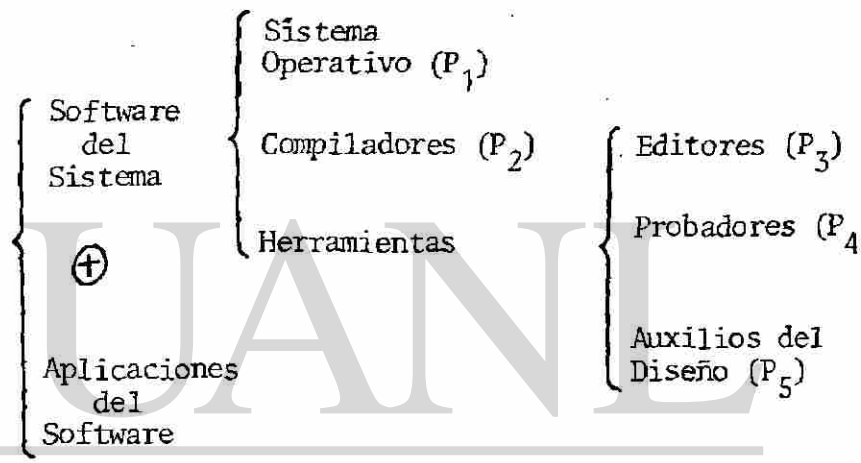
DIAGRAMAS WARNIER

El diagrama Warnier es un acercamiento diferente para la representación de la jerarquía de la información. Como el diagrama de bloques jerárquicos, el diagrama Warnier representa la información como una estructura de datos tipo árbol acostado. Pero, el diagrama Warnier también presenta descripciones alternativas.

La organización lógica de la información puede ser indicada con el diagrama Warnier. Esto es, una naturaleza repetitiva de una categoría específica de información o cantidad puede ser especificada. Ocurrencias condicionales de información dentro de una categoría puede ser mostrada. Porque las limitaciones repetitivas y condicionales son importantes para la especificación del procedimiento software, el diagrama Warnier puede ser convertido a la descripción del diseño soft-

ware con poca dificultad.

En el diagrama los paréntesis ( ) son utilizados -- para diferenciar los niveles de jerarquía de información. Todos los nombres contenidos dentro de los paréntesis corresponden a categorías de información o cantidades. El símbolo de OR exclusivo (⊕) indica ocurrencias condicionales de una categoría o cantidad y la notación entre paréntesis en seguida al nombre indica el número de repeticiones que ocurren en la estructura.



Dadas las convenciones para la notación, el diagrama - Warnier en la figura puede ser leído en la siguiente manera: una entrada de un producto Software consiste de información del sistema - Software y de aplicaciones Software. Donde existen P_1 sistemas operativos, P_2 compiladores y una subcategoría de herramientas Software. Bajo esta subcategoría contienen P_3 editores, P_4 probadores y P_5 auxiliares de diseño.

REQUERIMIENTOS DE BASE DE DATOS

Análisis de requerimientos para la base de datos incorpora tareas que son idénticas al análisis de requerimientos software. El contacto extensivo con el usuario es necesario, la identificación de funciones e interfases es esencial, especificación del flujo, estructura y asociatividad de la información es requerida y un documento formal de requerimientos se debe de desarrollar.

Una discusión completa del análisis de base de datos se pueden encostrar escritos por Martin, Wieder Hold o Cardenas.

CARACTERISTICAS DE BASE DE DATOS

El término base de datos se a convertido en una de muchas palabras claves del campo de computación. Existen muchas definiciones muy elegantes, pero definiremos a la base de datos como " una colección de información organizada en tal modo que facilita el acceso, análisis y los reportes".

La base de datos contiene entidades de información que se relacionan a travéz de la organización y asociación. La arquitectura lógica de la base de datos es definido por un esquema que representa las definiciones de las relaciones entre entidades de información. La arquitectura física de la base de datos depende en la configuración - - hardware. Pero, el esquema (descripción lógica) y la organización (- descripción física) se deben afinar para satisfacer los requerimientos funcionales y ejecucionales para el acceso, análisis y los reportes.

Un gran número de sistemas administrativos de base de datos (- DBMS) están disponibles para su compra. Pero no existe un estandard - industrial, el reporte del CODASYL DTBG 1971 puede eventualmente convertirse en la base de DBMS futuros.

ESPECIFICACION DE REQUERIMIENTOS DEL SOFTWARE

El producto que se desarrollo como parte del análisis de requerimientos es la especificación de requerimientos del software. La especificación extiende el objetivo (definido en la planeación del software), estableciendo una descripción completa de la información, una descripción funcional, un criterio de validación apropiado y otros - datos pertinentes a los requerimientos. El siguiente bosquejo se puede utilizar como un marco de referencia para las especificaciones:

1.- Introducción
2.- Descripción de la Información
    a) Diagramas de flujo de datos
    b) Representación de la estructura de datos
    c) Diccionario de datos
    d) Descripción de las interfases del sistema
    e) Interfases internas
3.- Descripción funcional
    a) Funciones
    b) Narración de la ejecución
    c) Limitaciones del diseño
4.- Criterios de Validación
    a) Límites de ejecución
    b) Tipos de pruebas

c) Respuesta esperada del software

d) Consideraciones especiales

5.- Bibliografía

6.- Apéndice

La introducción se enuncian las metas y los objetivos del software, describiéndolos en el contexto del sistema. La descripción de la información provee una descripción detallada del problema que el software tiene que solucionar. El flujo y la estructura de información son documentadas. Interfases hardware, software y humanas son descritas por elementos externos del sistema y funciones internas del software.

Los detalles de procedimiento para cada función requeridos para resolver el problema son descritos en la descripción funcional. Una narración del proceso se provee para cada función, las limitaciones del diseño son enunciadas y justificadas, uno o más diagramas de bloques son incluídas para representar gráficamente la estructura generalmente la estructura general del software y la interrelación entre las funciones del software y otros elementos del sistema.

Los criterios de validación actúa como una revisión implícita de la información y los requerimientos funcionales. Es esencial que se le de atención y tiempo a esta sección.

La bibliografía contiene referencias de todos los documentos que se relacionan al software. Estos incluyen otra documentación de la fase de planeación , referencias, técnicas literatura del vende--

dor, y estándares. El apéndice contiene la información que suplementa a la especificación. En el apéndice se presenta también; datos tabulares, descripción detallada de los algoritmos, mapas, gráficas y otro material.

Cuando los requerimientos para el software interactivo con el hombre son desarrollados, es a veces práctico preparar un manual preliminar del usuario como un suplemento al documento de requerimientos. El manual tiene dos propósitos:

1.- La preparación del manual forza al analista a ver el software en la perspectiva del usuario. Por lo tanto, consideración temprano es dada a la interfase humana.
2.- El usuario puede revisar el documento que describe la interfase hombre-máquina explícitamente.

El manual preliminar del usuario se presenta en el software como una caja negra. Esto es, se enfatiza en las entradas del usuario y las salidas resultante. El manual puede servir como una herramienta invaluable para descubrir problemas en la interfase hombre-máquina.

HERRAMIENTAS PARA EL ANALISIS DE REQUERIMIENTOS

Las herramientas se han desarrollado para proveer de un juego de procedimientos que guían al analista a travez de las especificaciones de los requerimientos. Existen muchas clases de herramientas, pero solo consideramos dos: las herramientas predominantes manuales y las herramientas predominantes automatizadas.

Las herramientas predominantes automatizadas, los requerimientos pueden ser descritos como un lenguaje de especificación que combinan - indecadores de palabras claves con un lenguaje natural narrativo. El - lenguaje de especificación es alimentado al procesador que produce -- las especificaciones de los requerimientos y más importante, un juego de reportes diagnósticos de la consistencia y organización de las especificaciones . Metodología ingenieril de requerimientos software - - (SREM) y lenguaje de estatutos del problema / análisis de los estatutos del problema (PSL/PSA) son herramientas automatizadas representativas de esta categoría.

S A D T

El SADT es una técnica de análisis y diseño del sistema que ha - sido ampliamente utilizado como una herramienta de definición del sistema, análisis de requerimientos, y diseño del software y del sistema. Consiste de métodos que permiten al analista descomponer las funciones del software (o del sistema), la notación gráfica, el diagrama de actividades del SADT, donde comunica las relaciones de información a una función dentro del software, y la guía de control del proyecto para la aplicación de la metodología.

Utilizando el SADT, el analista desarrolla un modelo que promete las jerarquías definidas en el diagrama de actividades. La metodología del SADT aprovecha las herramientas técnicas y una organización -- bien definida, en la cual, las herramientas son aplicadas. Las revisiones y requerimientos importantes son especificados, permitiendo validación de comunicación entre el desarrollador y el usuario.

HERRAMIENTAS AUTOMATIZADAS

Un número de herramientas automatizadas para la especificación de los requerimientos se han propuesto a travez de la última década. Un acceso a la automatización se ha precipitado por la dificultad en la validación de consistencia y lo completo de la descripción de sistemas manuales. Las herramientas automatizadas están en su infancia, y como maduran, pueden convertirse en una herramienta suplemental para todo análisis.

S R E M

La herramienta automatizada para el análisis de requerimientos utiliza el lenguaje de estatutos de requerimientos (RSL) para describir, los elementos, atributos, relaciones y estructuras. Los elementos (en la terminología SREM) comprende un juego de objetivos y conceptos que se utilizan para desarrollar las especificaciones de los requerimientos. Relaciones entre los objetivos son especificados como parte del RSL y los atributos son utilizados para modificar o calificar los elementos. Las estructuras son utilizadas para describir el flujo de información. Estas premicias del RSL son combinadas con una narración informativa para formar en detalle las especificaciones de los requerimientos.

El SREM aplica el sistema de validación y requerimientos ingenieriles (REVS) . El software del REVS utiliza una combinación de reportes y gráficas para estudiar el flujo de información a travez del sistema y simular interrelaciones dinámicas entre los elementos.

Como el SADT , el SREM incorpora un procedimiento que guían al analista a travez del paso de requerimientos. Estos procedimientos - incluyen:

1.- Traducción. Una actividad que transforma los requerimientos ini-- ciales descritos en la especificación del sistema en un juego más detallado de las descripciones de los datos y pasos de proceso.

2.- *Descomposición. Una actividad que evalua la información en la in*terdase al elemento software y resulta en un juego completo de re querimientos funcionales.

3.- Asignación. Una actividad que considera accesos alternativos a - los requerimientos que han sido establecidos.

4.- Demostración de Factibilidad Analítica. Una actividad que trata - de simular el procesamiento de requerimientos críticos para deter minar la factibilidad.

P S L / P S A

PSL/PSA fue desarrollado por el proyecto ISDOS en la Universidad de Michigan y es parte de un sistema mas grande llamado, diseño auxiliar por computadora y herramienta de análisis para la especificación (CADSAT) . El PSL/PSA provee al analista con las siguientes capacidades;

1.- Descripción de sistemas de información, sin importar el área de - aplicación.

2.- Creación de una base de datos conteniendo descriptores para el sistema de información.

3.- Adición, borrar o modificar de descriptores.

4.- La producción de documentación formateada y varios reportes de - la especificación.

La estructura del modelo PSL es desarrollado con la utilización de descriptores para el flujo de información, estructura del -- sistema, estructura de datos, derivación de datos, el tamaño y volúmen del sistema, dinámicas y propiedades del sistema y administra- - ción del proyecto.

Una vez que se complete la descripción del PSL, el PSA es - invocado. El PSA produce un número de reportes que incluyen un registro de todas modificaciones hechas a la especificación de la base de datos, reportes de referencia que presentan la información en la base de datos en formatos variados, reportes sumarios que proveen - información a la administración del proyecto , y reportes de análi-- sis que evaluan las características de la base de datos.

El PSL/PSA proveen de los siguientes beneficios:

1.- Calidad de documentación es mejorada a travez de estandarización.

2.- Coordinación entre analistas es mejorada porque la base de datos está disponible para todos .

3.- Omisiones e inconsistencias son descubiertas con mayor facilidad a travez referencias cruzadas en mapas y reportes.

4.- El impacto de modificaciones son fácilmante rasteables.

5.- Costos de mantenimiento para las especificaciones son reducidas.

El sistema CADSAT es representativo de los trabajos y metas oara las herramientas automatizadas como PSL/PSA y SREM. Herramien--

tas de análisis de requerimientos serán acoplados con las técnicas de diseño y otras herramientas para establecer un sistema de desarrollo - software.

CAPITULO V

EL PROCESO DE DISEÑO SOFTWARE :

El diseño es el primer paso en la fase de desarrollo para cualquier producto ó sistema. Puede ser definido como "El proceso de Aplicación de varias Técnicas y principios para el propósito de definir un dispositivo, un proceso ó un sistema consuficiente detalle para permitir su realización física".

La meta del Diseñador, es producir un modelo - ó representación de una entidad que después será construída. El - proceso, por la cuál, el modelo es desarrollado combina lo si- -- guiente : Intuición y juicio basado en la experiencia en la construcción de entidades similares, un juego de principios que guían el modo en que el modelo se desenvuelve, un juego de criterios -- que habilitan a la virtud a ser juzgado y un proceso de iteraciones que permite llegar a la presentación final del diseño.

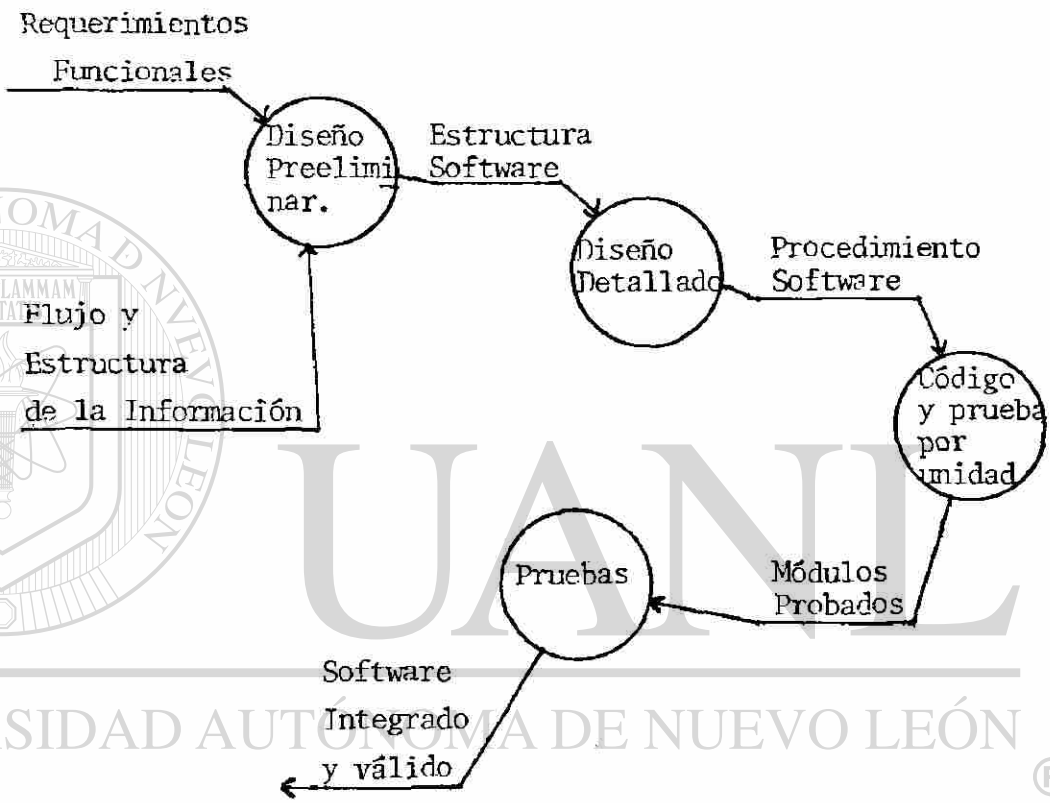
LA FASE DE DESARROLLO :

La fase central del ciclo de vida del Software es el desarrollo. Una vez que se establecieron los requerimientos Software, la fase de desarrollo está compuesta de cuatro pasos -- distictivos :

1. Diseño preeliminar
2. Diseño detallado
3. Codificación
4. Pruebas

Cada paso transforma la información en una forma que resulta en un Software válido.

El flujo de información durante la fase de desarrollo se ilustra en la siguiente figura.



Los requerimientos Software y el flujo estructura de la información alimentan el paso de diseño preeliminar. Con el uso de uno de varias metodologías de diseño, una estructura -- Software es desarrollada. La estructura Software también llamada-Arquitectura Software, define las relaciones entre los elementos principales del programa. El diseño detallado transforma los ele-

mentos estructurales en una descripción de procedimiento del Software. El código fuente es generado y pruebas preeliminares se conducen durante el paso de codificación y pruebas. - Integración detallada y pruebas de validación son ejecutadas en el último paso de la fase de desarrollo.

La fase de desarrollo absorbe cuando menos - el 75% del costo del nuevo Software. Aquí es donde hacemos - las decisiones que afectará el éxito de la implementación -- Software, e igualmente importante, la facilidad de manteni - miento del Software.

EL PROCESO DE DISEÑO :

El Diseño del Software es un proceso a tra - vés del cuál los requerimientos son traducidos en una representación del Software. Inicialmente la representación muestra un punto de vista holistico del Software. Refinamiento - subsecuente nos lleva una representación del diseño que está muy cercano al código fuente en el detalle de procedimiento.

Para evaluar la calidad de la representación del diseño, debemos establecer el criterio para el buen diseño. La siguiente guía nos muestran ciertos criterios:

1. Un diseño debe exhibir una organización jerár - quica que hace uso inteligente del control en - tre los elementos del Software.

2. El diseño debe ser modular, el Software debe --

ser partido lógicamente en elementos que ejecutan una función ó subfunción específica.

3. El diseño debe de conducir a módulos (sub-rutinas o procedimientos) que exhiben características independientes funcionales.

4. El diseño se debe derivar utilizando un método repetitivo que es accionado por la información obtenida durante el análisis de requerimientos Software.

Las características antes mencionadas del buen diseño no son realizables por oportunidad. El proceso de diseño alienta el buen diseño a través de metodologías sistemáticas y revisiones.

DOCUMENTACION DEL DISEÑO :

La documentación del diseño se evoluciona en el mismo sentido del esfuerzo técnico asociado con el diseño. Las primeras versiones de la especificación del diseño se concentraban en la Arquitectura del Software, mientras versiones posteriores representaban en detalle cada elementos Software.

La especificación del diseño sirve como propósito doble, provee de una guía a la implementación del Software (código) y pruebas y asistir al que mantiene el Software después de que se haya entregado. La especificación puede tener cambios considerables durante el ciclo de vida. Entonces, es-

esencial, controlar y revisar la documentación del diseño en - - cada paso de la fase de desarrollo.

El perfil del documento que continúa, puede ser - utilizado como modelo para las especificaciones del diseño. Cada sección contiene párrafos numerados que direccionan diferentes - aspectos de la representación del diseño.

1.0 Objetivo

1.1. Objetivo del sistema y el rol del Software como - elemento del sistema.

1.2. Interfaces Hardware, Software y Humanas

1.3. Funciones principales del Software

1.4. Base de datos definido externamente

1.5. Limitaciones principales del diseño

2.0 Documentos de Referencia.

2.1. Documentación existente del Software

2.2. Documentación del sistema

2.3. Documentos del vendedor (Hardware ó Software)

2.4. Referencias técnicas

3.0 Descripción del Diseño

3.1. Descripción de los datos

3.1.1. Revisión del flujo de la información

3.1.2. Revisión de la estructura de la información

3.2. Estructura derivada del Software

3.3. Interfaces dentro de la estructura

4.0 Módulos

Para cada Módulo:

4.1. Narración del proceso

4.2. Descripción de la Interfase

4.3. Descripción del lenguaje de diseño

4.4. Módulos utilizados

4.5. Organización de los datos

4.6. Comentarios

5.0 Estructura de Archivos y Datos Globales

5.1. Estructura de archivos externos

5.1.1. Estructura lógica

5.1.2. Descripción lógica del registro

5.1.3. Métodos de acceso

5.2. Datos globales

5.3. Referencias cruzadas de archivos y datos

6.0 Requerimientos de Referencias Cruzadas

7.0 Provisiones de Pruebas

7.1. Guías de pruebas

7.2. Estrategia de integración

7.3. Consideraciones especiales

8.0 Empaquetar

8.1. Provisiones especiales de incrustamiento del programa.

8.2. Consideraciones de transferencia

9.0 Notas Especiales

10.0 Apéndices.

El perfil de la documentación presenta una descripción completa del Diseño Software. Las secciones numeradas delas especificaciones del diseño son completadas como el diseñador refina su representación del Software.

El objetivo general del esfuerzo de diseño es descrita en la sección 1.0 de este perfil. Mucha de la información contenida en esta sección se derivó de los documentos de especificación del Sistema y de la Fase de Planeación del Software. Referencias específicas apoyando la documentación son hechas en la sección 2.0.

En la sección 3.0, la descripción del diseño, es completado como parte del diseño preliminar. Como se ha notado, el diseño es forzado por la información, esto es, el flujo y/o estructura de los datos dictarán la arquitectura del Software. En esta sección los diagramas de flujo de datos ó diagramas de estructura, desarrolladas durante el análisis de requerimientos, son afinidas y utilizados para derivar la estructura del Software. Porque el flujo de información está disponible, las descripciones de las Interfases pueden ser desarrolladas para loselementos del Software.

En las secciones 4.0 y 5.0, el diseño preeliminar evoluciona al diseño detallado. Los módulos, elementos direccionables separadamente, tales como sub-rutinas, funciones ó procedimientos, son descritos inicialmente en un lenguaje narrativo. La narración del proceso explica la función procedual del módu-

lo. Después, una herramienta del diseño detallado es utilizada para traducir la narración en una descripción estructural.

La descripción de la organización de los datos es contenida en la sección S.O. Las estructuras de los Archivos se mantienen en un medio secundario de almacenaje, son descri - tos durante el diseño preeliminar, datos globales son asignados, y una referencia cruzada que asocia los módulos individuales a archivos ó datos globales son estbalecidos.

En la sección 6.0 contiene las referencias cruza das de requerimientos. El propósito de esta matriz de referen - cias cruzadas es de establecer los requerimientos funcionales - (listados en la columna izquierda) son satisfechas por el dise- ño Software e indican cuál módulo (listados a través de la fila superior) son críticos para la implementación de requerimientos específicos.

En el primer estado en el desarrollo de la do- cumentación de prueba son contenidas en la sección 7.0 del docu mento de diseño. Una vez, que la estructura del Software e in - terfases sean establecidas, se puede desarrollar las guías para pruebas de módulos individuales y la integración del paquete -- por completo. Pero una especificación detallada del procedimien to de prueba no sea completada hasta después en la fase de desa rrollo, el desarrollo de estrategias de prueba y consideracio - nes especiales de prueba (Hardware especial ó simulación del- - Software) son desarrolladas como evoluciona el diseño de espe - cificaciones.

Limitaciones del diseño, como limitaciones de la - memoria física o la necesidad para alta ejecución, pueden dictar requerimientos especiales de asemblaje ó empaquetarse, del Software. Consideraciones especiales causados por la necesidad de incrustación de programas, administración de la memoria virtual, procesamiento a alta velocidad, u otros factores que pueden causar modificaciones en el diseño derivado del flujo de - información.

Las secciones 9.0 y 10.0 contienen datos suplementarios. Descripciones Algorítmicas, Procedimientos Alternos, - Datos Tabulares, Extractos de otros documentos y otra información relevante son presentados como una nota especial ó como - un apéndice separado.

REVISIONES DEL DISEÑO :

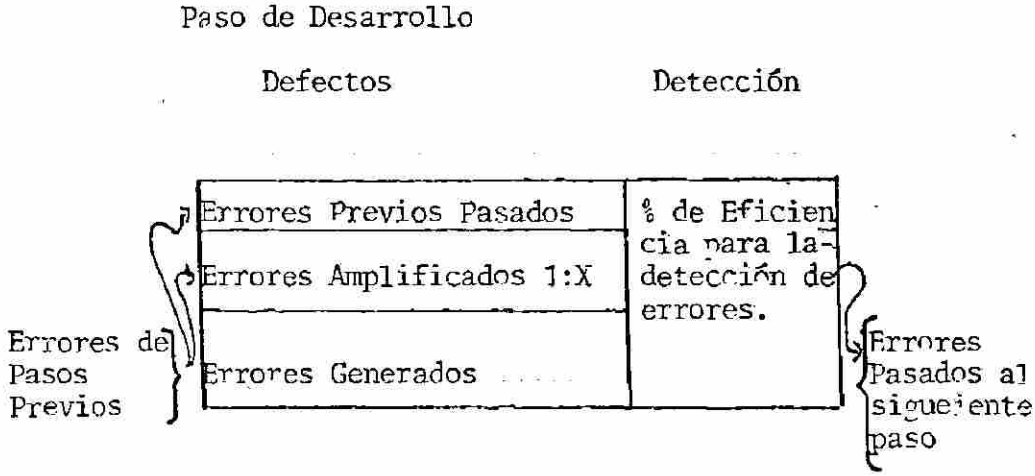
El flujo del proceso de diseño del Software son -- precisados con las revisiones. Una revisión bien planeada es - tan importante para el diseño del Software como los métodos -- técnicos de diseño. Dos filosofías de revisión de diseño coexisten y son aplicadas en diferentes puntos del proceso del diseño. Las filosofías residen en planos opuestos del espectro - de formalidad. En un plano, requiere una revisión formal con - dispositivos preparados cuidadosamente, público invitado, y -- una agenda planeada. En el otro plano, se conducen juntas el - vapor con varios compañeros para discutir la eficacia del diseño (revisión informal). En lugar de seleccionar un plano del - espectro de formalidad como un acceso exclusivo al diseño, el grado de formalidad puede ser confeccionada para la organiza -

ción de diseño y el tiempo que es conducida la revisión.

CONSIDERACIONES COSTO-BENEFICIO :

El beneficio obvio de las revisiones del diseño es descubrir los defectos Software, para que el defecto sea corregido antes de la codificación, pruebas y entrega. Un número de estudios industriales (TRW, Nippon Electric, Mitre Co.) indican que el paso de diseño del Software introduce entre un 50% - a un 65% de todos los errores (defectos) durante la fase de desarrollo del ciclo de vida del Software. Pero, técnicas efectivas de revisión pueden descubrir un gran porcentaje de éstos -- errores y reducir substancialmente el costo de los pasos subsecuentes en las fases de desarrollo y mantenimiento.

El modelo de amplificación de defectos puede ser utilizado para ilustrar la generación y detección de errores durante el diseño preeliminar, diseño de tallado, y pasos de codificación del proceso Software. El modelo esquemáticamente ilustrado en la siguiente figura. Un rectángulo representa el paso - de desarrollo del Software. Durante este paso, los errores pueden ser generados sin advertencia. La revisión puede fracasar en descubrir los errores generados y errores de pasos previos, resultando en una acumulación de errores que se pasan. En algunos casos, los errores previos son amplificados (por un factor X) -- por el trabajo corriente. Las subdivisiones representan cada una de éstas características y el porcentaje de eficiencia de detección de errores.



Para conducir revisiones, el desarrollador - - debe gastar tiempo, esfuerzo y dinero. En los resultados de va - rias Industrias han encontrado el síndrone de "pagar ahora ó - pagar mucho más después ". Las revisiones del diseño y código- proveen un costo de beneficio demostrable. Las revisiones se deben de conducir.

CRITERIOS DE REVISIONES DEL DISEÑO :

El Diseño del Software es revisado por representantes de : Administración, Desarrollo Técnico, Usuario, -- Asegurador de Calidad y Soporte del Software. Cada participante revisará el Diseño con su perspectiva, criterios selectos de revisión serán de importancia para cada participante :

1. Rastreabilidad.- ¿El diseño direcciona todas las facetas de la especificación de requerimientos del Soft - ware?

¿Es cada elemento del Software rastreable a un requerimiento específico?

2. Riesgo.- ¿La implementación del diseño requerirá un alto riesgo, es el diseño ejecutable sin una ruptura tecnológica?

3. Practibilidad.- ¿Es el diseño una solución práctica al problema identificado en los requerimientos?

4. Mantenibilidad.- ¿El diseño desarrollado guiará al fácil mantenimiento del sistema?

5. Calidad.- ¿El diseño exhibe características cualitativas del buen Software?

6. Interfaces. ¿Se han definido adecuadamente las interfaces externas e internas ?

7. Claridad Técnica.- ¿Es el diseño expresado en una manera que fácilmente sea traducible al código?

8. Alternativas.- ¿Se han considerado las alternativas del diseño? ¿Qué criterios se utilizaron para seleccionar la última elección?

9. Limitaciones.- ¿Son las limitaciones realísticas y consistentes con los requerimientos?

10. Intereses Especiales.- ¿Es el diseño del Software, probable, consistente con otros elementos del sistema y bien documentado?

Cada participante en la revisión del diseño puede enfatizar en uno o más de los criterios mencionados. Pero, una revisión completa es completada solo después de haber examinado todos los criterios.

REVISIONES FORMALES :

Las revisiones del Software formales normalmente son conducidos para evaluar la estructura y las interfases establecidas por el Software. Revisiones de este tipo son caracterizadas por preparación significante por el diseñador y el revisor, un gran número de revisores con varios grados de interés en el proyecto de desarrollo y visibilidad de la alta administración y técnica.

Como se ha notado, las revisiones formales del diseño son programadas e incluídas en el planeamiento Software. Cuando menos dos semanas antes de la fecha programada de revisión, la documentación del diseño es deserminada a todos los revisores. Desafortunadamente, muchos de los revisores no emplean mucho tiempo revisando la documentación. Por esta razón, el administrador del proyecto puede requerir con la respuesta escrita, forzando a cada revisor a dar atención al diseño. El proceso de revisión formal culmina con la presentación del diseño.

El formato de representación del diseño es establecido en una base de caso por caso. Los siguientes tópicos siempre son presentados:

* Identificación del Software y responsabilidad del diseño.
* Objetivos del diseño:
  *Visión de todos los elementos
  *Funciones principales del Software
  *Características de validación
* Requerimientos generales del Software
  *Modelo del sistema
  *Diagrama del flujo de datos
* Estructura Software
* Módulos presentados por función
* Estructura de los datos
* Requerimientos de la referencia cruzada
* Sumario

La revisión formal es normalmente conducida como parte del paso del diseño preeliminar, hay poco énfasis en el procedimiento del detalle dentro del Software.

El objetivo de la revisión formal debe ser evaluar el diseño, no los diseñadores. Porque son humanos los revisores y presentadores, una colisión de personalidades puede ocurrir . La revisión del diseño debe ser planeada y administrada en casi el mismo modo que los otros pasos del proceso Software.

Un límite de tiempo y programación debe ser establecido para la presentación y discusión de las funciones principales del Software, Una agenda se debe enforzar el administrador de revisión, quién debe asegurar que las siguientes -

guías se sigan :

1. Notas escritas de los comentarios de los revisores - deben ser tomadas. Es fácil para el desarrollador de olvidar (por error ó por diseño) muchas sugerencias- constructivas durante la revisión. Las notas pueden servir como lista de acción para el desarrollador y para la administración.

2. La revisión del diseño se deben emitir pero no necesariamente resolverse.

3. El número de participantes de revisión debe ser limitado.

REVISIONES INFORMALES :

Las revisiones informales son juntas al vapor a una - revisión más estructurada con compañeros. En esta sección consideraremos la conducción estructurada, ó sea una revisión planeada que es menos formal que la revisión formal, pero no menos -- efectiva.

Los participantes para esta revisión esencialmente -- los mismos que atendieron la revisión formal. Yourdon define las categorías genéricas para los participantes de la conducción estructurada, que incluyen:

* Un coordinador, quién es responsable de la planeación y actividades de organización.

* Un productor, quién ha desarrollado el diseño a revisar.
* Una secretaria, quién registra los eventos.
* Otros representativos desde mantenimiento, pruebas, y usuario.

En general, el número de representantes y la duración de la conducción estructurada son pequeños. Donde en la revisión formal puede involucrar de 8 a 12 personas, en la conducción estructurada se puede conducir con 2 o 3 participantes. En la revisión formal puede requerir días o semanas para terminar la revisión, en la conducción estructurada está limitada a una ó dos horas. El objetivo de la conducción estructurada, es de considerar un aspecto específico del Software en un ambiente informal.

INSPECCIONES :

Otro método a la revisión del diseño (y código) ha sido desarrollado e implementado por la IBM. La inspección se caracteriza por los elementos de ambas revisiones, formal e informal. La metodología de la inspección es bastante formal, y control son especificadas en avance. Pero, las personas involucradas y sus instrucciones tienen los atributos de la informalidad de pequeños grupos.

La metodología de inspección incorpora un número de revisiones (y actividades relacionadas). Un equipo de inspección, que se compone del moderador, el diseñador, el implementador un individuo especialmente capacitado, coordina a los otros miembros

del equipo y guía el progreso de la inspección.

El diseñador presenta su diseño al implementador del código (programador), quién es responsable de la traducción del diseño a un código fuente, y el probador, es el individuo quién conduce las pruebas del Software.

El proceso de inspección empieza con referencia general donde el diseñador presenta, el procedimiento lógico, caminos e interdependencias del diseño al equipo de inspección. El moderador puede hacer énfasis en ciertas áreas del diseño con una atención especial. Entonces, cada miembro del equipo de inspección, proparan individualmente un estudio del diseño para su inspección. El equipo de inspección completo conduce la inspección nombrando a un lector (que no es el diseñador) quién describe el diseño y como se implantará. Se hacen preguntas y los errores son descubiertos, pero no resueltos.

Después de la conclusión de la inspección, el moderador produce un reporte que especifica los errores descubiertos y los requerimientos que se necesitan retrabajar y revisiones futuras. El retrabajo es conducido por el diseñador y/o el implementador de código, y los problemas descubiertos son solucionados. Las futuras revisiones son conducidas para asegurar que cada error sea corregido.

C A P I T U L O VI

CONCEPTOS SOFTWARE :

Entre los muchos atributos que caracterizan la disciplina de Ingeniería es la habilidad para definir medidas significantes de un sistema. Lord Kelvin reconoció ésto cuando dijo :

"Cuando puedas medir lo que estás haciendo, sabes algo. Cuando no puedes, tu conocimiento es del tipo raquítico e insatisfactorio. Puede ser el inicio del conocimiento, pero escasamente has avanzado en tus pensamientos a un estado de ciencia".

Estamos al inicio de conocimiento, considerando las medidas para el Software computacional disponibles.

En este capítulo, consideraremos las medidas fundamentales que son aplicadas durante el diseño. Tales medidas son generalmente de una naturaleza cualitativa, pero no proveen una base accesible científica para la Ingeniería Software. Además, examinando los trabajos actuales en desarrollo de medidas cuantitativas para el Software. Esta investigación y experimentación puede algún día permitirnos avanzar al estado de ciencia.

CUALIDADES DEL BUEN SOFTWARE :

Jackson una autoridad en el diseño del Software, comenta con una visión pragmática: "El inicio de conocimiento para

el programador es reconocer la diferencia entre forzando al programa a trabajar y hacerlo bien".

El buen Software exhibe tres cualidades que lo hacen bien:

1. El Software trabaja según los requerimientos especificados, están rápido, eficiente y funcional como se requiera.

2. El Software es mantenible, puede ser diagnosticado y modificado sin gran dificultad.

3. El Software no es más que un código, es una configuración de documentos que nos aseguran que las dos primeras cualidades se realizen.

MODULARIDAD :

El concepto de modularidad en el Software se ha defendido por casi dos décadas. La estructura contiene modularidad, ésto es, el Software es dividido en elementos separadamente nombrados y direccionables, que son integrados para satisfacer los requerimientos del problema.

Se ha dicho que "la modularidad es atributo singular del Software que permite al programa sea administrada inteligentemente". El Software monolítico, un gran programa con un solo módulo, no es fácil para el lector reconocerlo. El número de caminos de control, las referencias, el número de variables, y -

la complejidad general, al entendimiento sería casi imposible.

Es importante hacer notar que el sistema pudo - haber sido diseñado modularmente, aunque su implementación sea monolítica. Hay situaciones (Software de tiempo real ó Software de microprocesadores) en el cuál, velocidades mínimas y memoria son introducidas por subprogramas (subrutinas y procedimientos) es inaceptable. En tales situaciones del Software puede y debe ser diseñado sin la filosofía de modularidad. El código puede ser desarrollado en línea. Pero, el código fuente - del programa no aparecerá modular a primera vista, pero la filosofía se ha mantenido, y el programa se proveerán los beneficios de un sistema modular.

ABSTRACCION :

Cuando consideramos una solución modular a cualquier problema, muchos niveles de abstracción se pueden proponer. En el nivel más alto de abstracción, una solución es mencionada en términos amplios, utilizando el lenguaje del ambiente del problema. En niveles medios de abstracción una orientación procedual es tomada. Terminología orientada al problema- es unido con la terminología orientada a la implementación es un esfuerzo para enunciar la solución.

Finalmente, en los niveles más bajos de abstracción, la solución es enunciada en el modo que pueda ser directamente implementada. Wasserman provee una definición útil:

"La noción psicológica de abstracción permite -

a uno concentrarse en el problema en algún nivel de generalización sin consideración a detalles irrelevantes de bajo nivel, - la utilización de abstracción también permite a uno trabajar -- con conceptos y términos que son familiares en el ambiente del problema sin tener que transformarlos a una estructura desconocida".

Cada paso en el proceso de Ingeniería Software es un refinamiento en el nivel de abstracción de la solución -- Software. Durante la definición del sistema, el Software es descrito como un elemento completo del sistema en contexto con todo el sistema. Durante la planeación del Software y análisis de requerimientos, la solución Software es enunciada en términos - que son familiares en el ambiente del problema. Si nos movemos- del diseño preeliminar al diseño de tallado, el nivel de abs -- tracción es reducido. Finalmente, en el nivel más bajo de abs - tracción es alcanzado cuando el código fuente es generado.

Los conceptos de refinamiento por pasos y modularidad son alineados cercamente con la abstracción, Como --- evoluciona el diseño Software, cada nivel de módulos en la es - tructura Software representa un refinamiento en el nivel de abstracción del Software. En realidad, la estructura factorizada distribuye los niveles de control y de decisión, esto es, niveles de abstracción.

INFORMACION ESCONDIDA :

El principio de Información Escondida sugiere

que los módulos deben ser caracterizados por desiciones de diseño que cada uno esconde la información de los otros. En otras - palabras, los módulos deben ser especificados y diseñados para que la información (datos ó procedimiento) contenidos dentro del módulo sean inaccesibles a otros módulos que no necesitan de tal información.

El término esconder implica que la efectividad - modular puede alcanzarse definiendo un juego de módulos independientes que se comunican con otros con solo la información que - es necesaria para obtener la función Software. La abstracción -- ayuda a definir las entidades de procedimiento (información) que contiene el Software.

La utilización de información escondida como un criterio de diseño para sistemas modulares provee de grandes beneficios cuando las modificaciones son requeridas durante las - pruebas, y después, durante el mantenimiento Software. Porque - la mayoría de datos y procedimientos son escondidas de otras -- partes del Software, errores inadvertentes son introducidos durante la modificación son menos probables de propagarse a otras localizaciones dentro del Software.

TIPOS DE MODULOS :

La abstracción y la información escondida son - utilizadas para definir los módulos dentro de la estructura Software. Ambos de éstos atributos deben ser traducidas a un semblante operacional modular, que se caracterizan por tiempo historial

de incorporación, mecanismos de activación y patrones de con - trol.

El tiempo historial de incorporación se refie re al tiempo en el cuál el módulo es incluído en la descrip -- ción del lenguaje fuente del Software.

Dos mecanismos de activación se encuentran - - convencionalmente, un módulo es invocado por una referencia - - (estatuto Call) pero, en aplicaciones de tiempo real, un módulo puede ser invocado por el estatuto interrup, esto es, un cuento exterior causa una discontinuidad en el procesamiento que resul ta en pasar el control a otro módulo. Los mecanismos de activa- ción son importantes porque pueden afectar la estructura del -- Software.

Los patrones de control del módulo describe la forma en la cuál es ejecutada internamente. Módulos convenciona les tienen una entrada y una salida y son ejecutadas secuencial mente como parte de una tarea del usuario.

Patrones de control más sofisticados a veces - se requieren pero, el módulo se puede utilizar por más de una - tarea a la vez.

Dentro de la estructura Software, un módulo - - puede ser categorizado de la siguiente forma:

* Un módulo secuencial, que es llamado y ejecutado sin interrupciones aparentes por la aplicación del Software.

* Un módulo incremental, puede ser interrumpido antes de la terminación por la aplicación y subsecuentemente, reiniciado en el punto de interrupción.

* Un módulo paralelo, ejecuta simultáneamente con otro módulo en un ambiente multiprocesador.

Los módulos secuenciales son los más convencionales, y se caracterizan por subprogramas, subrutinas, funciones ó procedimientos. Los módulos incrementales, también llamados corutinas, mantienen la dirección de entrada que permite al módulo reiniciar en el punto de interrupción.

Módulos paralelos, a veces llamadas conrutinas, son encontrados cuando la comunicación de alta velocidad demanda dos o más CPO's trabajando en paralelo.

Un control jerárquico típico (estructura factorizada) puede que no se encuentren, cuando se utilizen corutinas o con rutinas. Tales estructuras no jerárquicas ó homólogas requieren de accesos especiales de diseño que están en las primeras fases de desarrollo.

INDEPENDENCIA MODULAR :

El concepto de independencia modular es una excrecencia directa de modularidad y los conceptos de abstracción e información escondida. Parnas y Wirth aluden a las técni

.cas de refinamiento que engrandecen la independencia modular. Los trabajos de Stevens solidifican este concepto.

La independencia modular es lograda desarrollando módulos con una sola función y una aversión a una interacción - - excesiva con otros módulos. Dicho de otra forma, queremos diseñar el Software para que cada módulo direccione una subfunción específica de requerimientos y que tenga interfases simples cuando es - visto de otra parte de la estructura Software.

Software con modularidad efectiva, o sea, módulos independientes, son más fáciles de desarrollar porque la función- puede ser dividida, y las interfaces simplificadas. Módulos independientes son más fáciles de mantener y probar porque los efec - tos secundarios causados por el diseño y modificaciones del código son limitadas, se reduce la propagación de errores y la inserción de módulos es posible. Como resúmen, la independencia modular es la llave de un buen diseño y el diseño es la llave de la calidad Software.

La independencia es medida utilizando dos criterios cualitativos: cohesión y acoplamiento. La cohesión es la medida de fuerza funcional relativa de un módulo. El acoplamiento - es la medida de interdependencia relativa entre los módulos.

COHESION :

La cohesión es una extensión natural del concepto de información escondida. Un módulo cohesivo ejecuta una sola ta - rea dentro del procedimiento Software, requiriendo poca interac --

ción con otros procedimientos Software, siendo ejecutados en - otras partes del programa. Simplemente el módulo cohesivo debe - (idealmente) hacer una sola cosa.

La cohesión se puede representar como una escala no lineal. Siempre se quiere obtener una alta cohesión, pero, casi siempre se obtiene una media cohesión que es aceptable.

Un módulo que ejecuta un juego de tareas que -- vagamente se relacionan entre sí, se le llama cohesión coinciden tal. Un módulo que ejecuta las tareas que lógicamente se relacio nan se le conoce por cohesión lógica. Cuando un módulo contiene- tareas que se relacionan por la acción que se deben ejecutar en- la misma cantidad de tiempo, el módulo exhibe una cohesión tempo ral.

Cuando los elementos de procesamiento de un módulo están relacionados y deben de ejecutarse en cierto orden -- existe una cohesión de procedimiento (ó procedual). Cuando todos los elementos de procesamiento se concentran en una área de la - estructura de datos, la cohesión communicacional esta presente. - Cuando el elemento de procesamiento es relacionado a la misma -- función y deben ejecutarse en secuencia, entonces, se presenta - la cohesión secuencial. En el nivel más elevado de cohesión, es tá la cohesión funcional, la cuál, hace una sola función a la -- vez.

En la práctica, no es necesario determinar el preciso nivel de cohesión. Pero es importante tratar de lograr - un alto nivel de cohesión y reconocer un bajo cohesión para que el diseño pueda ser modificado para alcanzar una mayor indepen - dencia modular.

ACOPLAMIENTO :

El acoplamiento es una medida de interconección entre módulos en la estructura Software. La acoplación depende - de la complejidad de las interfases entre módulos, en el punto - en el cuál, hace su entrada ó referencia el módulo, en la cuál,- los datos pasan a través de la interfase.

En el diseño del Software se procurará por el - nivel más bajo posible de acoplamiento. La conección simple en - tre módulos resulta en un Software más fácil de entender y menos probable a que tenga un efecto de rizo causado cuando los erro - res ocurren en una dirección y se propagan a través del sistema.

Cuando se presenta un argumento simple, o sea, un dato simple es pasado, cuando existe una correspondencia uno a uno, el acoplamiento es bajo, acoplamiento de datos se exhibe en esta porción de la estructura. Una variación de este acoplamiento, se llama acoplamiento de estampa, se encuentra cuando - una porción de la estructura de datos se pasa a través de una - interface del módulo.

En niveles moderados de acoplamiento se caracteriza por el pasaje del control entre módulos, acoplamiento de control, el control es pasado por emdio de una bandera, en la - cuál, las decisiones son hechas por un módulo sobordinado o super ordinado.

Niveles relativamente altos ocurren cuando los módulos reciben información de un ambiente exterior. Por ejemplo,

acoplamientos de Entrada/Salida de módulos a periféricos específicos, protocolos de formatos y comunicación. El acoplamiento - externo es esencial pero se debe limitar a un número limitado - de módulos dentro de la estructura. El acoplamiento Common ocurre cuando un número de módulos hacen referencia al área global de datos.

El nivel más adecuado de acoplamiento es el - acoplamiento de contenido, ocurre cuando un módulo hace uso de- datos ó información de control mantenidas dentro de los límites de otro módulo. También ocurre cuando se ramifica a otro módulo. Este tipo de acoplamiento se debe de evitar.

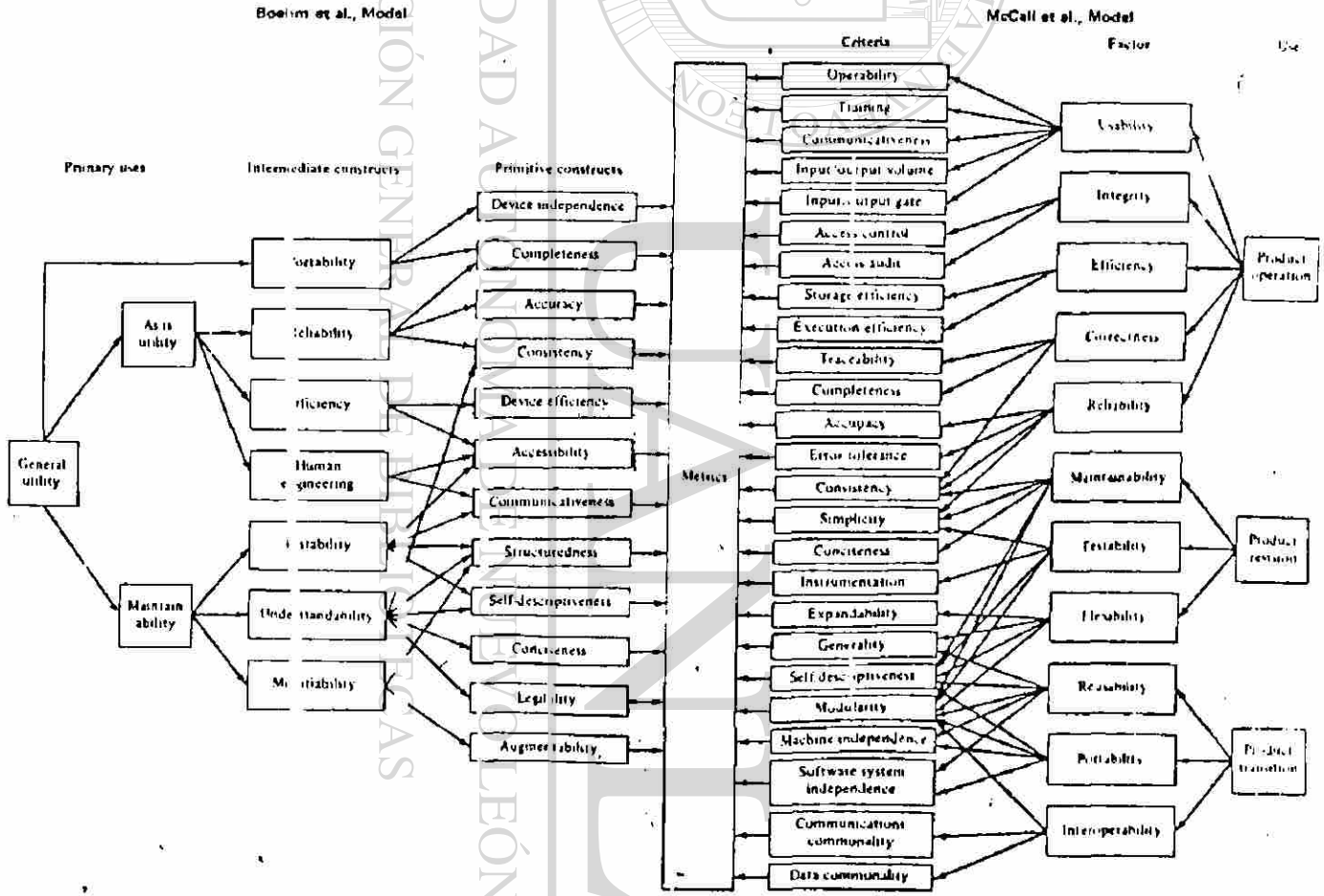
Los modos de acoplamiento anteriormente discutidos ocurren porque las decisiones del diseño hechas cuando -- la estructura fue desarrollada. Las variantes de acoplamiento - externa se pueden introducir durante la codificación.

MEDIDAS DEL SOFTWARE :

Las medidas que se han discutido anteriormente son cualitativas y no tienen una base matemática formal. Una relación entre conceptos tales como la independencia modular y calidad Software se presume lógicamente que existen.

En una investigación de medidas Software, Curtis describe tres usos principales para las medidas del Software:

1. Herramientas de información para la administración.
2. Medidas de calidad del Software.



(Source: William Curtis, Management and Experimentation in Software Engineering, Proceedings of the IEEE, Vol. 68 No. 9, September 1980, p. 1147. Reprinted with permission of)

3. Retro alimentación para el Ingeniero Software.

Herramientas de información para la administración se aplican en la fase de planeación del ciclo de vida del Software. Productividad de datos provee uno de varias medidas cuantitativas que se pueden aplicar.

Las medidas de calidad del Software son más elusivas. La figura presenta una síntesis del trabajo conducido por Behem y McCall que intentan proveer de medidas útiles para la calidad del Software. La construcción primitiva de Boehem y el criterio de McCall se pueden reducir a un juego de medidas que se puedan medir en una forma cuantitativa. La jerarquía de calidad se muestra en ambos lados de la figura es útil como medio de evaluación cualitativo.

Trabajos adicionales sobre medidas del Software se han concentrado en la complejidad del Software, estructuras de control, interconección (independencia modular) y la teoría de la ciencia de Software de Halstead. La complejidad se categoriza en términos de complejidad computacional, una especifica ción formal de la estructura de algoritmos, eficiencia, aplicación y complejidad psicológica (una medida de factores humanos que afectan al desarrollo del Software). Estructuras de control Software utiliza la teoría gráfica para medir las características interiores de procedimiento, tales como el número de ramificaciones, ó caminos de procesamiento. En las secciones siguientes, dos métodos cuantitativos de medidas Software se presentarán.

CIENCIA SOFTWARE DE HALSTEAD :

La teoría de ciencia de Software de Halstead es

probablemente la mejor conocida y la más estudiada, compuesta de medidas de complejidad Software. La ciencia de Software propone las primeras leyes analíticas para el Software computacional.

La ciencia del Software es una área relativamente nueva de investigación que aisgna leyes cuantitativas al desarrollo del Software. La teoría de Halstead se deriva de una asunción fundamental : "El cerebro humano sigue un juego de reglas rígidas en el desarrollo de algoritmos que lo que se percata" . La ciencia del Software utiliza un juego de medidas primitivas que se pueden derivar después que el código sea generado o estimado una vez que el diseño sea completado.

n_1 es el número de operadores distintos que aparecen en el programa.

n_2 es el número de operandos que aparecen en el programa.

N_j número total de ocurrencias del operador

N_2 número total de ocurrencias del operando.

Halstead utiliza las medidas primitivas para desarrollar expresiones para la longitud total del programa, el volúmen mínimo potencial para el algoritmo, al volúmen actual (el número de bits requeridos para especificar el programa), el nivel del lenguaje (una constante para un lenguaje dado), y otras características, tales como, el esfuerzo del desarrollo, tiempo de desarrollo, y el número proyectado de errores del Software.

Halstead muestra que la longitud N puede ser es-

timada como :

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

Y el volúmen del programa se puede definir como :

$$V = N \log_2 (n_1+n_2)$$

Se debe hacer notar que el volúmen del programa - variará con el lenguaje de programación utilizado y representa- el volúmen de información en bits requeridos para especificar - el programa.

Teóricamente, el volúmen mínimo debe existir para el algoritmo en particular. Halstead define la relación de volúmen L como la relación del volúmen en su form más compacta de - un programa al volúmen actual del programa. O sea, L siempre debe de ser menor que la unidad. En términos primitivos de medi - das, la relación de volúmen se puede expresar como :

$$L = (2/n_1) * (n_2/N_2)$$

Halstead propuso que cada lenguaje puede ser categorizado por el nivel de lenguaje, 1, el cuál variaría entre- lenguajes. Teóricamente el nivel de lenguaje es una constante - para el lenguaje dado, pero en recientes trabajos indican que - el nivel de lenguaje es una función del lenguaje y del programador. Los siguientes valores del nivel de lenguaje han sido derivados empíricamente para los lenguajes comúnes :

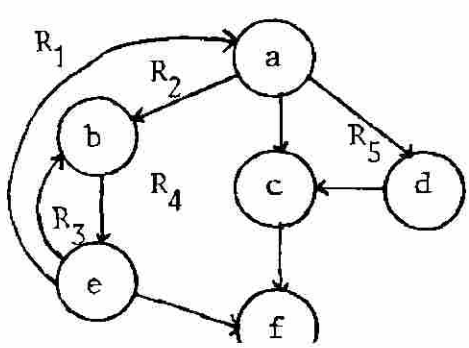
Lenguaje	1 Promedio
Inglés (prosa)	2.16
PL/1	1.53
ALGOL/68	2.12
Fortran	1.14
Asemblador	0.88

Aparenta que el nivel de lenguaje implica un nivel - de abstracción en la especificación del procedimiento. Lengua - jes de alto nivel permite la especificación del código en un nivel alto de abstracción, que el lenguaje asemblador.

Las medidas cuantitativas propuestas por Halstead -- tienen una aplicación limitada en la práctica. Pero, la ciencia del Software muestra una promesa como una herramienta cuantitativa para la confiabilidad del Software, estimación del esfuerzo de mantenimiento del Software, y como una medida formal de - complejidad y modularidad.

MEDIDA DE COMPLEJIDAD DE Mc CABE :

La medida de complejidad propuesta por Thomas Mc -- Cabe se basa en el flujo de control representando al programa. Una gráfica del programa ilustrada en la figura es utilizada para presentar el flujo. Cada letra circundada representa una tarea de procesamiento (una o más líneas de código), el flujo de- control (ramificaciones) se representan conectando las flechas. Entonces, para la gráfica G, la tarea a puede ser seguida por - las tareas b, c, ó d, dependiendo de las condiciones probadas - en a. La tarea b siempre le seguirá la tarea e y ambos pueden - ejecutarse como parte de a, en la doble bifurcación (las flechas curvas).



V(G) = 5

Mc Cabe define la medida de complejidad del -- Software, que rebasa en la complejidad ciclomática de la gráfica del programa para un módulo. Una técnica que puede ser utilizada para calcular la medida de complejidad ciclomática, VCG), es determinar el número de regiones en la gráfica plana (para - mayor información ver a Bondy y Murty). La región puede ser -- descrita informalmente como una área encerrada en el plano. El- número de regiones se calcula, contando todas las áreas encerradas y el área sin límite de exterior a la gráfica. En la gráfica de la figura tiene cinco regiones y su medida de complejidad ciclomática V(G) = 5.

Porque el número de regiones se incrementa con el número de caminos de decisiones y las bifurcaciones, la medida de Mc Cabe provee una medida cuantitativa para probar la dificultad e indicar la confiabilidad. En estudios experimentales indican una relación entre la medida de Mc Cabe y el número de errores en el código fuente existentes también en el tiempo requerido para encontrar y corregir tales errores.

Mc Cabe también contiende que la V(G) puede - ser utilizada para proveer una indicación cuantitativa del tamaño máximo del módulo. Los datos coleccionados de un número de - proyectos, se ha encontrado que V(G) = 10 aparenta ser el límite superior para el tamaño del módulo. Cuando la V(G) es mayor, se convierte en extremadamente difícil de probar adecuadamente el módulo.

Las medidas de Halstead y McCabe son representantes del- creciente acercamiento cuantitativo de medida del Software. Herramientas automatizadas que asisten en la computación de ambas medidas se han desarrollado. La medida McCabe puede ser aplicado después de completar el diseño detallado del proceso Software, la medida Halstead requiere del código. Por lo tanto, la complejidad - ciclomática ofrece una herramienta de evaluación para probar y la confiabilidad que puede convertirse en un criterio para la revisión de un módulo.

HEURISTICAS DEL DISEÑO :

Los conceptos Software introducidos en este capítulo forman la base para las heurísticas del diseño que pueden aplicarse- indiferente de la metodología del diseño específico siendo utilizado. En esta sección examinaremos un número importante de heuristicas del diseño. Acoplando éstas guías con nociones, tales como, modularidad e independencia, se proveerá una fundación para métodos de diseño que se discutirán en los capítulos siguientes.

1. Evaluar la estructura preeliminar del Software para reducir el acoplamiento y mejorar la cohesión. Una vez -- que la estructura del Software ha sido desarrollada, -- los módulos pueden ser explotados ó implotados con la - idea de mejorar la independencia modular, La descrip -- ción de procesamiento de cada módulo es examinado para determinar si un componente de proceso puede ser explotado a dos o más módulos y redefinidos como un módulo cohesivo separado. Cuando se espera un nivel de acoplamiento alto, los módulos se pueden, a veces, implotar -

para reducir el pasaje de control, referencia a datos globales y complejidad de la interfase.

2. Tratar de minimizar estructuras tipo abanico abierto-procurando el tipo de abanico cerrado como se incre - menta la profundidad.

3. Obtener el efecto del objetivo del módulo dentro del objetivo de control de ese módulo. El efecto del objetivo del módulo m es definido como todos los otros módulos son afectados por la decisión hecha por el - módulo m. El objetivo de control del módulo m es to - dos los módulos que son subordinados y últimamente subordinado al módulo m.

4. Evaluar las interfases del módulo para reducir complejidad y mejorar la consistencia. La complejidad de la interfase modular es la principal causa de errores -- Software. Las interfases se deben diseñar para pasar información simple y debe de ser consistente con la - función del módulo.

5. Definir módulos cuya función sea predecible, para evitar los módulos que sean sobre restrictivos. Un módulo predicable puede ser tratado como una caja negra, esto es, los mismos datos externos se producirán sin importar los detalles de procesamiento interno. Módulos que tienen memoria interna pueden ser inpredicable, a menos que, se tome cuidado en la utilización.

Un módulo que restringue el procesamiento a una sola subfunción exhibe una alta cohesión y es favorable para el diseñador. Pero, un módulo que arbitrariamente restringue el tamaño de la estructura de datos local, opciones -- dentro del flujo de control ó varios modos de interface exterior requerirán invariablemente mantenimiento para remover tales restricciones.

6. Procurar por módulos de una entrada, una salida, evitando conecciones patológicas. Esta guía del diseño advierte sobre el uso del acoplamiento de contenido. El Software es más fácil de entender y más fácil de mantener cuando los módulos son direccionados de arriba y su salida por abajo. El término conección patológica se refiere a las ramificaciones o referencias hechas al centro del módulo.

7. Paquetes de Software basado en limitaciones del diseño y requerimientos de portabilidad. El empaquetamiento alude las técnicas utilizadas para asemblar el Software a un ambiente específico de procesamiento ó enviar el Software a una localidad remota. Las limitaciones del diseño a veces dictan un programa que sobre encime en memoria. Cuando esto ocurre, la estructura del diseño puede ser reorganizada a grupos modulares por el grado de repetición, frecuencia de acceso, e intérvalos entre llamadas. Además, módulos -- opcionales, o de uso esporádico pueden ser separados en la estructura para que se puedan sobreencimar eficientemente.

8. Seleccionar el tamaño de cada módulo para que su independencia se mantenga. En casi todas las discusiones sobre modularidad tiende a preguntarse: "¿cuál es el tamaño correcto para un módulo?" En la respuesta se deben considerar tres características:

   1. El concepto de independencia modular
   2. La necesidad de reducir la complejidad
   3. La necesidad de recomendaciones cuantitativas.

   En casi todos los casos, un intento de lograr un alto nivel de cohesión, resultará en un módulo relativamente pequeño.

   Reduciendo el acoplamiento, la complejidad del módulo es reducida también.

C A P I T U L O VII

DISEÑO ORIENTADO AL FLUJO DE DATOS:

El Diseño ha sido descrito como un proceso de múltiples pasos, en la cuál, representaciones de la estructura y -- procedimiento son sintetizados de los requerimientos de información ésta descripción es extendida por Freeman:

"El Diseño es una actividad que compromete en ha - cer decisiones mayores, a veces de naturaleza estructural. Com - parte con la programación la inquietud de la abstracción de la - representación de información y secuencias de procesamiento, pero el nivel de detalle es muy diferente en los extremos. El Diseño construye coherencia, representaciones bien planeadas de programas que se concentran en las interrelaciones de las partes en un nivel elevado y las operaciones lógicas involucradas en los - niveles inferiores".

Las metodologías de diseño orientado al flujo de datos se presentará en este capítulo. El objetivo de este método es proveer de un acceso sistemático para el desarrollo de una estructura Software. un enfoque arquitectónico del Software y la - apuntalación del paso preeliminar del diseño.

DISEÑO Y FLUJO DE INFORMACION :

El Flujo de Información, es la principal consideración durante el paso de análisis de requerimientos. Iniciando- con un modelo de un sistema fundamental, la información puede --

ser representada como un flujo contínuo que sufre una serie de transformaciones (procesos) como evoluciona desde la entrada - a la salida. El diagrama de flujo de datos (DFD) es utilizada para representar el flujo de información. El diseño orientado- al flujo de datos define un número de mapas diferentes que - - transforman el flujo de información a una estructura Software.

El diseño orientado al flujo de datos tiene- un campo grande en las áreas de aplicación, porque todo Soft - ware puede ser representado por un diagrama de flujo de datos. Un método de diseño que utiliza el diagrama puede teóricamente aplicarse en cada esfuerzo de desarrollo Software.

Un acceso orientado al flujo de datos al diseño es particularmente fuerte cuando no existe una estructura formal de datos. Por ejemplo, aplicaciones de control por microprocesador, procedimientos de análisis numérica, control - de proceso, y otras aplicaciones que no requieren una estructura de datos sofisticado y son difíciles de modelar con el diseño orientado a la estructura de datos.

Una extensión al diseño orientado al flujo - de datos, llamado MASCOT, se adapta a aplicaciones de tiempo - real. Utilizando una representación del flujo de información - de procesos concurrentes, MASCOT provee una intercomunicación- de áreas de datos que permiten la definición de la comunica -- ción de interprocesos. Mucho énfasis en las aplicaciones de -- tiempo real debe de ser en las interfases entre varios proce - sos. MASCOT permite al diseñador especificar interfases y coor

denadas de comunicación con sincronización primitiva.

CONSIDERACIONES DEL PROCESO DE DISEÑO :

El diseño orientado al flujo de datos permite - una transición conveniente de la representación de información, - el diagrama de flujo de datos (DFD), contenidas en la especificación de requerimientos del Software, a una descripción del diseño preliminar de la estructura Software. La transición del flujo de información a la estructura es efectuado como parte de un proceso de cinco pasos :

1. La categoría del flujo de información es establecida
2. Las limitaciones del flujo son indicadas.
3. El DFD es mapeado en una Arquitectura Software
4. La jerarquía de control es definida por factorización.
5. La estructura resultante es reafinada por la utilización medidas y heuristicas del diseño.

FLUJO DE TRANSFORMACION :

La información entra al sistema a través de caminos que transforman los datos externos en una forma interna. El flujo a través de los caminos de entrada se llaman afferentes. En el núcleo del Software, ocurre la transición. Los datos de entrada son pasados por el centro de transformación y empiezan a moverse a través de los caminos que salen del Software.

El flujo en los caminos de salida se llaman effer entes. Cuando un segmento del DFD exhibe éstas características, el fluio de transformación está presente.

FLUJO DE TRANSACCION :

El flujo de información es a veces caracterizado - por un solo dato, llamado transacción. dispara otros flujos de- datos a través de un o de varios caminos. El flujo de transac - ción se caracteriza por los datos moviéndose a través de cami - nos de recepción que convierten la información externa en una - transacción. La transacción es evaluada, y en base a este valor, el flujo es iniciado a través de uno de muchos caminos de ac -- ción. El centro del flujo de información del cuál muchos cami - nos de acción brotan se llama el centro de transacción.

Se debe notar que en un DFD para un sistema grande los flujos de transformación y transacción pueden estar presente.

ANALISIS DE TRANSFORMACION :

El análisis de transformación es un juego de pasos de diseño que permite al DFD con características de flujo de -- transformación sea mapeado en una estructura predefinida del -- Software. Hay muchos ejemplos para el análisis de transformación que se han desarrollado para aplicaciones de procesamiento de - datos comerciales.

PASOS DEL DISEÑO :

Paso 1. Revisar el modelo fundamental del sieño. Este paso ini-

cia con la evaluación de la especificación del sistema y la especificación de los requerimientos del Software. Ambos documentos describen el flujo y la estructura de información en la interfase del Software.

Paso 2.- Revisar y refinar los diagramas del flujo de datos para el Software.

Paso 3.- Determinar si el DFD tiene características de transformación o de transacción. En general, el flujo de información dentro del sistema se puede representar como una transformación. Pero, cuando características obvias de transacción se encuentran.- un mapeo diferente del diseño se recomienda. En este paso, el -- diseñador selecciona la característica del flujo global basado - en la naturaleza predominante del DFD. Además, regiones locales del flujo de transformación ó de transacción son aisladas. Estos subfluios se pueden utilizar para refinar la estructura Software derivada de las características globales.

Paso 4.- Aislar el centro de transformación, especificando los - límites de flujo afferentes y efferentes.

Paso 5.- Ejecutar la factorización de primer nivel. La estructura del Software presenta una distribución del control de arriba hacia abajo. La factorización es un proceso que distribuye el -- control.

Cuando se encuentra un flujo de transformación, el DFD es mapeado en una estructura específica que provee con - trol para el flujo afferente (entrada),transformada, y efferente

(saliente) del procesamiento de información.

Paso 6.- Ejecutar la factorización de segundo nivel. La factorización de segundo nivel se efectúa mapeando las burbujas de transformación individuales del DFD en módulos apropiados de la estructura Software. Iniciando en el límite del centro de transformación y moviéndose hacia afuera a través de los caminos afferentes y después efferentes, las transformadas son mapeados en niveles subordinados de la estructura Software.

Paso 7.- Refinir el primer corte de la estructura Software, utilizando las medidas y heurísticas del diseño. El primer corte de la estructura Software puede ser refinida aplicando los conceptos de independencia modular. Los módulos son explotados ó implotados para producir una factorización sensible, buena cohesión, acoplamiento mínimo y más importantes una estructura que se puede implementar sin dificultad, pruebas sin confusión, y mantenimiento sin penas.

ANALISIS DE TRANSACCION :

El flujo -e información frecuentemente representa un dato que en su evaluación dispara flujos adicionales a través de uno de un número de caminos seleccionados.

PASOS DEL DISEÑO :

Los pasos del diseño para el análisis de transacción son similares y en algunos casos idénticos a los pasos para el análisis de transformación. La mayor diferencia cae en el mapeo DFD de la estructura del Software.

Paso 1. Revisar el modelo fundamental del sistema
Paso 2. Revisar y refinar el DFD para el Software
Paso 3. Determinar si el DFD tiene características de transformación ó de transacción.

Paso 4. Identificar el centro de transacción y las características de flujo para cada camino de acción. La localidad del centro de transacción se puede reconocer inmediatamente del DFD. El centro de transacción está en el origen de un número de caminos de información que fluyen radialmente de el. Los caminos de recepción y todos los caminos de acción se deben de aislar. Cada camino de acción se deben de evaluar sus características de flujo individual.

Paso 5.- Mapear el DFD en una estructura Software que sea tratable es mapeado en la estructura Software, que contiene una rama de recepción y una rama de despacho. La estructura para la rama de recepción es desarrollada en casi el mismo modo que el análisis de transformación. Inciando en el límite del centro de transacción, las burbujas a través del camino del flujo de recepción son mapeados en módulos. La estructura de la rama de despacho contiene el módulo despachador que controlan todos los módulos subordinados en una estructura que corresponde a las características del flujo específico.

Paso 6.- Factorizar y refinar la estructura de transacción y la estructura de cada camino de acción. Cada camino de acción del DFD tiene sus propias características de flujo de información. Como se ha notado, los subflujos de transformación ó transacción se pueden encontrar. La subestructura relacionada al camino de acción se desarrolla utilizando los pasos de diseño.

Paso 7.- Refinar el primer corte de la estructura del Software utilizando las medidas y heurísticas del diseño.

OPTIMIZACION DEL DISEÑO :

El Diseñador Software se debe concernir en desarrollar una representación del Software que concurren con los requerimientos funcionales y ejecucionales, y el mérito de aceptación basado en las medidas y heurísticas del diseño.

El refinamiento de la estructura Software durante las primeras fases del diseño se debe de animar. Representaciones alternativas pueden ser derivadas, refinadas y evaluadas para el mejor acceso. Este acceso para optimizar es uno de los beneficios derivados por el desarrollo de la representación de la Arquitectura Software.

Es importante notar que una simplicidad estructural refleja a veces elegancia y eficiencia. La optimización del diseño debe procurar por el menor número de módulos que sea consistente con la modularidad efectiva y la estructura menos compleja de datos que adecuadamente sirven los requerimientos de información.

Para aplicaciones de tiempo crítico, puede ser necesario de optimizar durante el diseño detallado y posiblemente, durante la codificación. El desarrollador del Software debe notar que, relativamente un porcentaje pequeño (entre 10 y 20%) del programa, a veces es responsable por un gran porcentaje (entre el 60 y 80%) de todo el tiempo de procesamiento.

Es razonable proporner los siguientes criterios para el Software de tiempo crítico :

1. Desarrollar y refinar la estructura sin concernimiento para la optimización del tiempo crítico.

2. Durante el diseño detallado, seleccionar los módulos que se sospecha que consumen en mucho tiempo y cuidadosamente desarrollar procedimientos (algoritmos) para la eficiencia en el tiempo.

3. Codificar en un lenguaje de alto nivel.

4. Instruccionar al Software aislar los módulos que tienen una sobre utilización de procesador.

5. Si es necesario, rediseñar ó recodificar en un lenguaie máquina para mejorar la eficiencia.

Estos criterios dictan la siguiente frase :

"Hazlo trabajar y luego hazlo más rápido"

C A P I T U L O VIII

DISEÑO ORIENTADO A LA ESTRUCTURA DE DATOS :

La relación íntima entre el Software y los datos tiene sus vestigios en el origen de la computación. El concepto original atrás de la computadora almacenadora de programas era que los programas se pueden visualizar como datos y datos -- interpretados como programas. La estructura de la información, - llamada estructura de datos, se ha demostrado que tiene un im -- pacto importante en la complejidad y eficiencia de los algorit - mos diseñados para procesar información.

Como han evolucionado los métodos de diseño a través de la década pasada, una escuela de pensamiento afirma - que "la identificación de una estructura de datos inherente es vital, y la estructura de datos de entrada y salida pueden ser utilizados para derivar la estructura de un programa y algunos detalles". En muchas áreas de aplicación existe una estructura de información jerárquica propia. Datos de entrada, información almacenada internamente (base de datos) y datos de salida pue - den tener cada uno una estructura única. El diseño orientado a la estructura de datos (DOED) hace uso de éstas estructuras como fundación para el desarrollo del Software.

EL DISEÑO Y LA ESTRUCTURA DE DATOS :

La estructura de datos afecta al diseño en - - sus aspectos estructural y procedual del Software. Datos repetitivos son siempre procesados con el Software que tiene el control sobre las facilidades de repetición, datos alternativos (información que puede o no estar presente) precipitar al Software con - elementos condicionales de procesamiento, una organización jerarquica de datos, frecuentemente tiene un parecido al Software que utiliza los datos. La estructura de la información es un excelente pronosticador para la estructura del Software.

El DOED puede aplicarse exitosamente, en aplicaciones que tengan una bien definida estructura jerárquica de informa - ción. Los ejemplos típicos incluyen:

Aplicaciones administrativas y financieras. Las entradas y salidas tienen una estructura propia, la utilización de - una base de datos jerárquico es común.

Aplicaciones de Sistemas.- La estructura de datos para sistemas de operación, es compuesta de muchas tablas, archivos y listados que tienen una estructura bien definida.

Aplicaciones CAD/CAM. Sistemas de diseño auxiliado -- por computadora y manufactura auxiliada por computadora requieren de una estructura de datos sofisticada para el almacenaje - de información, traducción y procesamiento.

CONSIDERACIONES DEL PROCESO DE DISEÑO :

El análisis de requerimientos del Software sigue siendo la fundación para el diseño orientado a la estructura de datos. La descripción de la estructura de información contenida -- en las especificaciones de requerimientos del Software prefigura la arquitectura del Software que se ha de desarrollar durante el diseño. Cada método de diseño provee un juego de reglas que habilita al diseñador transformar la estructura de datos a una representación del Software.

1. Las características de la estructura de datos es evaluada
2. Los datos son representados en términos de sus formas elementales. tales como, secuencia, selección y repetición.
3. La representación de la estructura de datos es mapeada en un control jerárquico para el Software.
4. La jerarquía del Software es refinado utilizando las guías definidas como parte del método.

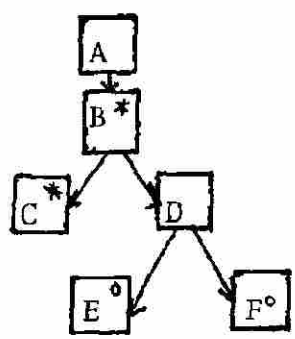
5. Una descripción procedual del Software es desarrollado. Una división entre los pasos del diseño preeliminar y detallado no son evidentes en los métodos orientados a la estructura de datos. Jackson y Warnier, ambos se - - mueven rápidamente a una representación procedual.

LA METODOLOGIA JACKSON :

La esencia de la metodología Jackson se puede mencionar, con las palabras de su desarrollador, Michael Jackson: "Los problemas se deben descomponer en estructuras jerárquicas de las partes que se pueden representar en tres formas estructurales". Las tres formas estructurales que menciona Jackson son secuencia - les, condicionales y repetitivas.

Jackson ha desarrollado una notación de estructura de datos que se asemeja al diagrama jerárquico de estructura de datos. Además, la metodología propone un juego de mapas ó procedi -- mientos de transformación. Es a través de éstos procedimientos que el método se puede adaptar a las variaciones en la estructura de datos de entrada y salida.

Una representación simple de la notación de estructura de datos se muestra en la siguiente figura :



El diagrama jerárquico. La colección de datos A es - - compuesta de múltiples ocurrencias (denotado por un asterisco) de la subestructura de datos B. La subestructura B incluye las ocurrencias múltiples de C y otra subestructura D, que contiene los datos E ó F (datos alternativos denotados por °).

La representación del diagrama en bloques de Jackson de la jerarquía de información pueden ser aplicadas a estructuras de entrada, de salida ó a base de datos con una facilidad igual.

Jackson contiende que procesando la jerarquía para los mapas Software directamente de una estructura de datos de entrada y/o salida. Desafortunadamente el, les llama estructuras - - jerárquicas de procesamiento, causando confusión en las definiciones anteriores del libro.

Al desigual que el diseño orientado al flujo de datos, los bloques de un procesamiento jerárquico no necesariamente delinean los módulos. Jackson toma una idea mezclada de modulari - dad, anticipando problemas potenciales durante la integración -- y mantenimiento del Software.

La metodología Jackson deriva un procesamiento jerár - quico como una representación primaria del diseño. Una extensión natural del procesamiento jerárquico es una representación procedual de un programa en un pseudocódigo, una notación tipo lenguaje de programación.

La metodología Jackson apoya un número de técn - nicos suplementarias que engrandecen su aplicabilidad y enriquecen el diseño en general. Varias de éstas técnicas suplementa -- rias se describen como sigue :

Validación de datos y procesamiento de errores. El problema con datos erróneos ó datos fuera de secuencia es difícil de resolver en un diseño orientado a la estructura de da - tos. El diagrama de estructura de datos no representa los datos erróneos porque tal información no existe en la estructura. En - realidad, los datos erróneos ocurren anticipadamente. Jackson se refiere : "La estructura del programa debe de tener reglas consistentes para la alocación de responsabilidad para probar la validez de los datos". Las siguientes reglas son:

* Cada especificación del módulo debe tener definido su - rango de datos válidos.

* Cada módulo es diseñado debe asumir que los datos dados son válidos.

* Si el módulo B es subordinado el módulo A, entonces el - módulo A es responsable de asegurar que los datos pasados al módulo B son válidos.

Estas reglas no pueden derivar directamente de la estructura de datos. A diferencia de los métodos orientados al - - fluio de datos que transforman el procesamiento de errores en el - DFD, Jackson no mapea tales características del diseño. En algu -

nos casos, pruebas para validación de datos y/o procesamiento - de errores se pueden incluír.

Retrotraceabilidad.- Jackson reconoce que durante una ejecución del programa, un camino de procesamiento se escoge a veces en - evidencia pobre o no existente. Esto es, asumimos el camino de - ser correcto y retrotraceamos cuando la evidencia nos indica -- que el camino no es el correcto. Para ayudar en el diseño en ta- les situaciones de retrotraceabilidad. Jackson propone tres nue- vas construcciones para el pseudocódigo procedual.

El término "Posit" se utiliza para indicar -- que el procesamiento ocurrirá en la base de la hipótesis de lo - correcto. El estado del programa es almacenado en la entrada a - una construcción "Posit". El término "Quit" indica que la hipóte sis falla y el control es pasado a la construcción "admit" que - produce un camino alternativo de procesamiento.

Choque de Esturcturas.-En muchas aplicaciones la información de entrada no tiene o casi poca correspondencia estructural a la in- formación de salida. El término utilizado por Jackson para ésta - situación es choque de estructuras y propone una serie de movi -- mientos del diseño detallado para corregirlo:

1. Las características de la estructura de entrada son definidas utilizando la notación del diagrama de estructura - de datos.

2. El Software es diseñado para descomponer la es -

tructura de datos de entrada en elementos que forman la estructura intermedia de datos.

3. La estructura de datos de salida es descrita.

4. El Software es diseñado para construír la estructura de salida de la estructura intermedia.

Para choques de estructura complicados, el diseñadorpuede sintetizar un número de estructuras intermedias de datos. Cada diseño resultante debe ser integrado para formar un programa.

CONSTRUCCION LOGICA DE PROGRAMAS :

La construcción lógica de programas (LCP), inicia con una representación de la estructura de datos, luego a una representación formal del procedimiento y culmina con métodos sistemáticos para la generación del pseudo código, verificación y optimización.

Warnier desarrolló este método al asumir que "Los programas pueden construirse lógicamente y verificados rigurosamente utilizando las herramientas derivadas del estudio de informática.

El LCP se presenta como una serie de reglas y leyes que gobiernan la estructura de información y la organización resultante derivada del Software. El LCP se deriva de --

fundaciones teóricas y es claramente el método más rígido en el tratamiento de desarrollo y verificación del diseño.

El diseño LCP inicia con la especificación de las estructuras de entrada y salida con la utilización de -- los diagramas Warnier. Como otros métodos de diseño, una rigu - rosa evaluación de los requerimientos Software es el precursor de la derivación de la representación del Software. Warnier toma una visión clásica que los programas, como los datos y resultados son archivos de información. Entonces, el siguiente - paso de LCP es representar el procesamiento del Software con - los diagramas Warnier. La jerarquía de procesamiento para un - programa se deriva de la estructura de los datos de entrada.

La LCP intenta extender la metodología del diseño en un dominio que otros métodos evitan. Warnier ha desarrollado una técnica, llamada "organización detallada", en la cuál, un juego de instrucciones detallados pueden sistemáticamente desarrollarse de la organización lógica del programa.

Otros métodos de diseño establecen la fundación para la especificación de instrucciones detalladas, pero Warnier ha propuesto un proceso de paso por paso para derivar tales instrucciones. Warnier define los siguientes tipos o clases de instrucción :

* Entrada y preparación de entrada
* Ramificación y preeliminares de ramificaciones
* Cálculos

* Salida y preparación de salidas.

* Llamadas a subprogramas (módulos)

La organización detallada es desarrollada generando listados de instrucciones por tipo. La instrucción es escrita y correlacionada al bloque de procesamiento apropiado con indicación numérica.

Después de los listados de instrucción sean preparados, las instrucciones con el mismo identificador son agrupados y organizados en una secuencia de entrada -proceso- salida. La organización detallada provee al diseñador con una técnica para el desarrollo de la descripción detallada del diseño en una manera sistemática. La naturaleza no secuencial de este -- acceso puede parecer un poco raro. pero el método trabaja.

DISEÑO DE DATOS :

Los métodos de diseño orientado a la estructura de datos implican que la organización de los datos es una característica dominante en el diseño del Software. Los conceptos de información escondida y abtracciones de los datos proveen la fundación para el diseño de datos. La mayoría de los métodos de diseño se concentran en las consideraciones estructurales (jerárquicos) ó proceduales.

El proceso de diseño de datos es resumido por - -

Wasserman :

"La actividad primaria durante el diseño de datos identificados durante la fase de definición y especificación de requerimientos. El proceso de selección puede involucrar un análisis -- algorítmico de estructuras alternativas para determinar el diseño más eficiente o simplemente involucrar la utilización de un juego de módulos (un paquete) que proveen las operaciones deseadas en - la representación de un objeto.

Una actividad importante relacionada durante el diseño es identificar los módulos del programa que deban operar directa - mente sobre las estructuras lógicas de datos. De esta manera, el - objetivo del efecto de las decisiones del diseño de datos indivi - dual puede ser limitado.

Indiferentemente de las técnicas de diseño utilizadas, los datos bien diseñados puede guiar a una mejor estructura Soft - ware, independencia modular, y reducir la complejidad procedual.

Wasserman ha propuesto un juego de principios que pueden ser utilizados para especificar y diseñar los datos. Recono -- ciendo que el análisis de requerimientos y el diseño a veces se -- traslapan, consideraremos el siguiente juego de principios para la especificación de datos :

1. Los métodos sistemáticos de análisis aplicados al Software, también deben aplicarse a los datos. Las representaciones -- del flujo y estructura de datos (diagramas DFD, jerárquicas, ó Warnier) también deben de desarrollarse y revisadas, orga

nizaciones de datos alternativos se deben de considerar, y el impacto del diseño de datos en el diseño del Software - se debe evaluar. Una organización alternativa de datos nos puede guiar a mejor resultados.

2. Todas las estructuras de datos y las operaciones que se -- van a ejecutar se deben de identificar.

3. Un direccionario de datos se debe establecer y utilizado -- para definir el diseño de datos y Software. El direcciona - rio representa explícitamente la relación entre los datos y las limitaciones en los elementos de la estructura de -- datos.

4. Decisiones de diseño de datos de bajo nivel se deben diferir hasta tarde en el proceso de diseño.

5. - la representación de la estructura de datos debe ser conocida solo por los módulos que hacen uso directo de los datos contenidos en la estructura.

6. Una librería de estructuras de datos y operaciones útiles que pueden aplicarse a ellas deben de ser desarrolladas. - Las estructuras de datos y operaciones se deben ver como - un recurso para el diseño del Software. Solamente, los paquetes de subrotinas pueden reducir grandemente el tiempo para el Software de utilería. La librería de estructura de datos pueden reducir el esfuerzo de especificaciones y de-

diseñado para los datos.

7. El diseño Software y el lenguaje de programación deben de -- darle soporte a la especificación y realización de tipos -- abstractos de datos. La implementación y el diseño de estructura datos sofisticados se pueden hacer muy difícil si no -- hay medios para la especificación directa de las estructuras.

Los principios descritos forman una base para el diseño de datos que se pueden integrar al diseño orientado al flujo de -- datos y el diseño orientado a la estructura de datos. Como se ha definido, una definición clara de la información es esencial para el éxito del desarrollo del Software.

CAPITULO IX

HERRAMIENTAS DE DISEÑO DETALLADO :

La metodología propuesta en este libro se acerca a dos pasos distintos. El primero, el diseño preeliminar establece - la estructura modular del Software. La segunda, el paso del diseño detallado completa todo el detalle procedual necesario. En la culminación del segundo paso, debe de existir una representación del- diseño del cuál el código fuente puede ser derivado directa y sencillamente.

Durante el paso de diseño preeliminar, cada módulo es definido como parte del Software. Además, de las descripciones del diagrama de estructuras e interfases, un proceso narrativo- - es desarrollado para cada módulo. El proceso narrativo es la descripción en lenguaje natural de la función y ejecución de un módulo. Herramientas representando al diseño son requeridas para transformar la narración en una descripción estructurada precisa del -- procedimiento.

Las herramientas de diseño son aplicadas durante - cada paso del proceso de desarrollo del Software. Herramienta que representan los datos son utilizadas durante el análisis de requerimientos para la especificación del flujo o estructuras de información. Los requerimientos se pueden transforman en herramientas - para la representación de la estructura del Software. Finalmente,- herramientas para la especificación del detalle procedual para completar la descripción del diseño.

Las herramientas para la especificación del procedimiento, llamadas, herramientas del diseño detallado, se pueden categorizar de la siguiente forma:

Herramientas Gráficas : El detalle procedual se representa como una parte de un retrato, en la cuál, la construcción lógica toma formas pictoriales específicas.

Herramientas Tabulares: El detalle procedual se representa utilizando tablas que describe las acciones y las condiciones correspondientes ó alternativas de la información de entrada, procesamiento y salida.

Herramientas de Lenguaje:El detalle procedual se representa con una descripción en pseudo código que se asemeja a un lenguaje de programación.

Indiferentemente de la categoría, la herramienta de diseño debe de indicar el flujo de control función de procesamiento, organización de datos, e implementación del detalle.

CONSTRUCCIONES ESTRUCTURADAS :

Las fundaciones del diseño detallado inició en los '60 y se solidificaron por el trabajo de Edsgar Dijkstra y sus colegas. En los '60, Dijkstra y otros propusieron un juego de construcciones lógicas, del cuál, cualquier programa se podría formar. Las construcciones enfatizaban un dominio funcional de mantenimiento. Esto es, cada construcción tenía una es -

tructura lógica, entraban en el principio y salían por el final.

Las construcciones son secuenciales, condicionales y repetitivas. La secuencia implementa los pasos de procesamiento que son esenciales en la especificación de cualquier algoritmo. La condición provee la facilidad para seleccionar al procesamiento basado en la ocurrencía lógica. Y la repetición provee las bifureaciones. Estas tres construcciones son fundamental para la programación estructurada.

Las condiciones estructuradas fueron propuestas para limitar el diseño procedual del Software a un número más -- pequeño de operaciones predicables. La medida de complejidad de McCabe indica que la utilización de construcciones estructuradas reduce la complejidad del programa y engrandece la legibilidad, las pruebas y la mantención.

Cualquier programa, sin importar el área de --- aplicación o complejidad técnica, puede ser diseñado e implementado utilizando solo éstas tres construcciones estructurales.

Se debe notar, que el uso dogmático de éstas -- construcciones pueden, a veces, causar dificultades prácticas.

HERRAMIENTAS GRAFICAS DE DISEÑO :

"Un retrato, vale mil palabras", pero Carl MacHover dice, "Es más importante conocer cuál retrato y cuál es mil - palabras". No hay preguntas que las herramientas gráficas, ta-

les como, diagramas de flujo ó diagramas caja, proveen excelentes patrones pictóricos que describen al detalle procedual. Pero, si las herramientas gráficas son mal usadas, el mal retrato nos puede guiar el Software equivocado.

El diagrama de flujo es el método más utilizado para la representación del diseño del Software. Desafortunadamente, el método más abusado.

El diagrama de flujo es muy sencillo pictorialmente. Un rectángulo es utilizado para indicar un paso de procesamiento. Un diamante ó rombo representa una condición lógica y las flechas indican el flujo de control. La secuencia se representa como dos rectángulos de procesamiento conectados por una flecha o línea de control. La condición, tan bien designada como "if-then-else", es representado como un diamante de decisión, si es verdadero causa a la parte "then" a procesarse, y si es falso, invoca a la parte "else" a procesarse.

La repetición se representa utilizando dos formas distintas. El término "do while" prueba la condición y ejecuta la biforcación hasta que la condición sea verdadera. El término "repeat un til" ejecuta la bifurcación primero y luego prueba la condición y repite la condición hasta que falle la condición.

Los diagramas rectangulares o caja, evolucionaron del deseo de desarrollar una herramienta gráfica de diseño

que no permita la violación de las construcciones estructurales. Desarrollados por Nassi y Shneiderman y extendidos por Chapin, - los diagramas, también llamadas los diagramas de Nassi-Sheider - man o diagramas N-S o diagramas Chapin, tienen las siguientes -- características:

1. Dominio funcional, es bien definidoy claramente visi ble como una representación pictórica (objetivo de la construc ción específica).

2. Transferencia arbitraria del control es imposible.

3. El objetivo local y/o global de datos puede ser fácil mente determinados.

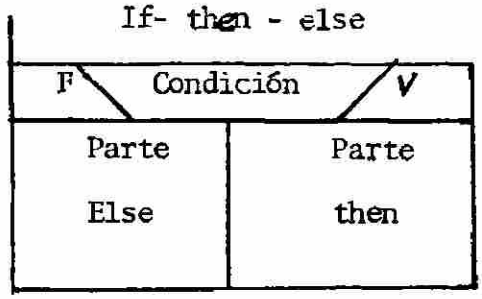
4. La revisión es fácil de representar.

El elemento fundamental del diagrama es el rectángulo. Para representar la secuencia, dos o más rectángulos se conectan de abajo a arriba. Para representar. "if-then-else" un rectángulo condicional es seguido por la parte "then"y por la -- parte "else" en rectángulos. La repetición se representa por un- patrón limitante que encierra el proceso que sea repetico (par - tes de "do-while" y "repeat-until".

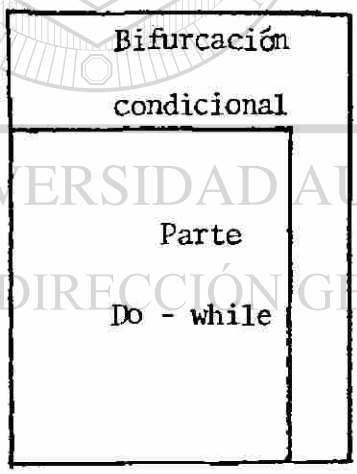
Finalmente, la selección es representada utilizando las formas gráficas mostradas en la siguiente figura:

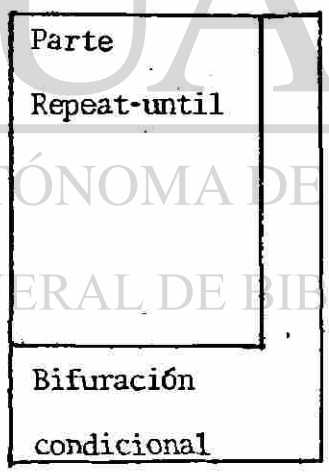
Secuencia

If- then - else

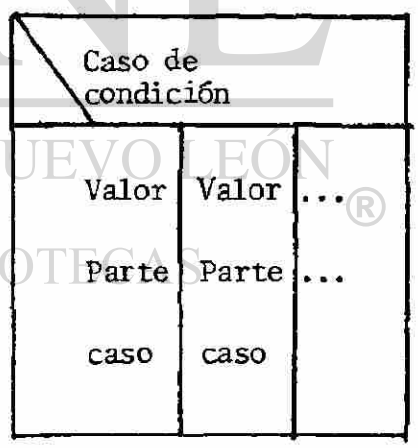


Repetición





Selección



Los diagramas caja pueden aparecer inusuales ó --ilegibles a un lector quién los encuentra por primera vez. En realidad, no es más complejo, en el sentido pictórico, que un diagrama de flujo. Un estatuto "Call" ó llamado a un módulo de subrutina puede representarse por un rectángulo con el nombre del módulo encerado en un óvalo.

Como un resúmen de esta técnica, considerando las palabras de Nassi y Shneiderman: "Los programadores quién primero - aprende a diseñar programas con éstos símbolos, nunca desarrollará malos hábitos del diseño procedual que otros notaciones de diagramas de flujo permiten. "El uso de construcciones estructurales como un estado de mente. Si una herramienta de diseño enforza solo las construcciones estructuradas, este modo de pensar es continuamente reenforzado y eventualmente se convierte en natural.

COMPARACION DE HERRAMIENTAS DE DISEÑO:

Cualquier comparasión de las herramientas de diseño debe de predecir en la aserción que cualquier herramienta para el diseño detallado, si es utilizado correctamente, puede ser un auxilio invaluable en el proceso de diseño, pero, la mejor herra -- mienta, si mal aplicada, es poco entendible. Con esta idea en la- mente, examinaremos los criterios que se pueden aplicar para comparar las herramientas.

Las herramientas del diseño detallado nos deben -- guiar a una representación procedual que sea fácil de entender - y revisar. Además, la herramienta debe clarecer la habilidad de codificar, para que el código se convierta en un producto del diseño. Finalmente la representación debe de mantenerse fácilmente para que el diseño siempre represente al programa correctamente.

Los siguientes atributos de las herramientas de diseño han sido establecidas en el contexto de las características generales descritas anteriormente.

Modularidad .- La herramienta de diseño (HD) debe dar soporte al desarrollo de módulos del Software y proveer los medios de especificación de la interface.

Simplicidad General.- La HD debe de ser relativamente fácil de aprender y utilizar y leer.

Facilidad de Edición.- El diseño detallado puede requerir alguna modificación durante el paso de diseño, pruebas y la fase de manutención del ciclo de vida del Software. La facilidad, con la cuál, la representación del diseño pueda ser editado -- pueda ayudar a facilitar cada paso del diseño.

Legibilidad Maquinal.- Hay una gran tendencia a la utilización de técnicas automatizadas para el desarrollo del Software. Una herramienta que pueda directamente dar entrada a un sistema de desarrollo basado en la computadora puede ofrecer grandes beneficios.

Mantenibilidad.- La mantención del Software es la fase más -- costosa.

Coacción Estructural.- La HD que enfuerze la utilización de solo las construcciones estructurales es una práctica del buen diseño.

Procesamiento Automático.- El diseño detallado contiene información que se puede procesar para dar al diseñador una percepción nueva o mejores para la calidad de un diseño.

Representación de Datos.- La habilidad para representar los datos locales y globales es un elemento esencial del diseño detallado.

Verificación de Lógica.- Verificación automática del diseño lógico es una meta durante las pruebas del Software .

Habilidad de Codificar.- El paso que sigue del diseño detallado es la codificación. Una herramienta que pueda convertirse fácilmente a un código fuente reduciendo el esfuerzo y errores.

C A P I T U L O X

LENGUAJES DE PROGRAMACION Y CODIFICACION :

Todos los pasos de la Ingeniería de Software presentados hasta este punto, son direccionados a un objetivo final : a traducir las representaciones del Software en una forma que pueda ser entendible por la computadora. Finalmente hemos llegado al paso de codificación, un proceso que transforma el diseño en un lenguaje de programación. La manera, en el cuál, ejecutamos la programación puede cambiar dramáticamente si la generación automática del código viniera a existir. Por este tiempo, codificaremos utilizando los lenguajes artificiales, tales como, FORTRAN, PASCAL, COBOL, Basic ó Lenguaje Assembler.

Cuando es considerado como un paso en el proceso de Ingeniería Software, la codificación se puede ver como una consecuencia del diseño. Pero, las características del lenguaje de programación y el estilo del código pueden afectar profundamente la calidad y la mantenibilidad del Software.

EL PROCESO DE TRADUCCION :

El paso de codificación traduce la representación del diseño detallado del Software en una realización del lenguaje de programación. El proceso de traducción continua cuando un compilador acepta el código fuente como entrada y produce un código objeto dependiente de la máquina como salida. El código fuente es retraducido en un código máquina, las instrucciones actuales son --

microcodificadas lógicamente en la unidad central de procesamiento (CPU).

El paso inicial de traducción, del diseño detallado al lenguaje de programación, es el interés primario en el contexto de la Ingeniería Software. El ruido puede entrar a la traducción en muchas formas. Interpretación inpropia del diseño detallado puede guiar a un código fuente erróneo. La complejidad ó restricciones del lenguaje de programación nos puede guiar a un código fuente convolucionado que es difícil de probar y mantener . Más sútilmente, las características del lenguaje de programación puede influenciar la manera en que pensamos, la propagación innecesaria de diseños de Software y estructuras de datos limitados. Las características del lenguaje tienen un fuerte impacto en la calidad y eficiencia de la traducción.

CARACTERISTICAS DEL LENGUAJE DE PROGRAMACION :

El Lenguaje de Programación es un vehículo de comunicación entre humanos y las computadoras. El proceso de codificación es una actividad humana. Como tal, las características psicológicas del lenguaje tienen un impacto importante de comunicación. Las características ingenieriles de un lenguaje tiene un fuerte impacto en el éxito del proyecto en desarrollo. Finalmente, las características técnicas del lenguaje pueden influenciar la calidad del diseño.

En su reciente libro Software Psychology, Ben Shneiderman, observó que el rol de Psicólogo Software es de "enfo

car el interés humano, tal como, la facilidad de uso, simplicidad en el aprendizaje, mejorar la confiabilidad, reducir la frecuencia de errores y engrandecer la satisfacción del usuario, pero -- manteniéndose cauteloso en la eficiencia de la máquina, capacidad Software y limitaciones del Hardware".

Un número de características psicológicas ocurren como resultado del diseño del lenguaje de programación. Estas características no son medibles en una forma cuantificable, pero, se reconoce su manifestación en todos los lenguajes de programación. Estas características son :

Uniformidad.- Indica el grado, en el cuál, el lenguaje utiliza notación concisa.

Ambiguedad .- La ambiguedad en el lenguaje de programación es percibido por el programador. El compilador siempre interpretará un estatuto en una sola forma, pero el ser humano puede interpretarlo diferente.

Compactibilidad.- La compactibilidad del lenguaje de programación es una indicación de la cantidad de información orientada al código que deba recordar la memoria humana. Entre los atributos del lenguaje que sirven para indoxar la compactibilidad son :

* El grado en el cuál el lenguaje da soporte a las -- construcciones estructuradas.
* Los tipos de palabras clave y abreviaciones que se pueden utilizar

* La variedad de tipos de datos y la omisión de - - características.

* El número de operadores aritméticos y lógicos.

* El número de funciones internas.

Las características de la memoria humana tienen un - fuerte impacto en el modo que utilizamos el lenguaje. La memoria - y el reconocimiento humano se puede dividir en los diminios sin estética y secuencial. La memoria sinestética nos permite recordar - y reconocer las cosas como un todo. La memoria secuencial provee - los medios para recordar cuál es el siguiente elemento en una se - cuencia. Cada una de éstas características de memoria afectan a -- las características del lenguaje de programación, llamadas, localidad y linearidad.

La localidad es una característica sin estética del lenguaje de programación. La localidad se engrandece cuando los - estatutos pueden combinarse en bloques, cuando la construcción estructurada puede ser implementada directamente y cuando el diseño y el código resultante tiene altamente modular y cohesivo.

Una visión de la Ingeniería Software de las carac - terísticas del lenguaje de programación se enfoca en las necesidades del proyecto específico. Aunque los requerimientos isotéricos para el código fuente pueda ser derivado, un juego general de - - características ingenieriles se pueden establecer :

1. Facilidad de traducción del diseño al código.
2. Eficiencia del compilador, esta habilidad de producir un código rápido y la utilización de poca memoria.
3. Portabilidad del código fuente, esta característica - se puede interpretar de tres formas diferentes :
   3.1. El código fuente puede ser transportado de procesador a procesador y de compilador a compilador con poco o sin modificación.
   3.2. El código fuente permanece sin cambios aunque -- cambie su ambiente.
   3.3. El código fuente puede ser integrado en diferentes paquetes Software con poca o sin modifica -- ción.
4. Disponibilidad de herramientas de desarrollo, pueden reducir el tiempo requerido para generar el código -- fuente y puede mejorar la calidad del código.
5. Mantenibilidad del código fuente tiene importancia -- crítica para todos los esfuerzos de desarrollo no triviales del Software.

Las características técnicas de los lenguajes de programación existen un gran número de tópicos que tienen un rango desde teórico a pragmático. Las características técnicas de - muchos lenguajes de programación pueden afectar los conceptos de diseño durante la implementación del diseño.

CLASES DE LENGUAJES :

Existen cientos de lenguajes de programación que

han sido aplicados a los esfuerzos de desarrollo. Pero, se pueden categorizar en tres tipos de lenguajes de alto nivel: Lenguajes de fundación, Lenguajes estructurados y Lenguajes especializados. Un Lenguaje, también, puede estar en dos categorías.

Los Lenguajes de fundación son caracterizados por su amplia utilización, enormes librerías de Software, y una amplia gama de aceptación y familiaridad. Hay poco debate que el FORTRAN, COBOL, ALGOL y Basic son Lenguajes de fundación por virtud de madurez y aceptación.

La versión original estandarizada del FORTRAN -66 proviò una herramienta poderosa para la solución de problemas computacionales, pero, carecía soporte directo de las construcciones estructuradas, tenía pobres tipos de datos, no había la facilidad para el manejo de "strings" y muchas otras deficiencias. El nuevo estandard del ANSI llamado FORTRAN-77 corrigió muchas de las deficiencias encontradas en versiones anteriores del Lenguaje.

El COBOL, como el FORTRAN, alcanzó su madurez y es aceptado como un Lenguaje estandard para aplicaciones de -- procesamiento comercial. Aunque, el Lenguaje es criticado por su falta de compactabilidad, tiene excelente tipos de datos, es auto documentado, y provee el soporte para un amplio rango de técnicas de procedimiento relevantes al procesamiento de datos administrativos.

El ALGOL es un Lenguaje pre estructural y -- ofrece un repertorio bastante rico de construcciones proceduales

y tipos de datos. La versión más comúnmente utilizada se llama ALGOL-60, ha sido extendida a una implementación más poderosa- ALGOL-68. Ambas versiones del lenguaje, soportan la noción de- estructuras de bloques, distribución dinámica de almacenamiento recursión y otras características que tienen gran influencia en los lenguajes estructurados.

El BASIC es un Lenguaje interpretativo que fue originalmente diseñado para enseñar a programar en un modo de tiempo compartido. Existen cientos de versiones del BASIC, - haciendo muy difícil de discutir los beneficios y deficiencias del lenguaje.

Los lenguajes estructurados se caracterizan por sus fuertes capacidades de estructuras proceduales y datos. Todos los Lenguajes de esta clase soportan directamente las - - construcciones estructuradas lógicas. El Lenguaje estructurado, ALGOL, sirvió como modelo para otros lenguajes en esta catego - ría. Sus descendientes, PL/1, PASCAL, C, y ADA, están siendo -- adoptados como lenguajes con un amplio potencial de aplicacio - nes.

El PL/1, el primer lenguaje de un espectro amplio. Esto es, fue desarrollado con un rango amplio de fac - ciones que lo habilita ser utilizado en muchas áreas de aplica ción. El PL/1 provee de soporte para aplicaciones científicas, ingenieriles y administrativos, mientras que, al mismo tiempo habilitando la especificación de estructura de datos sofistica das, multi programación, entradas y salidas complejas, procesa

miento de listados y muchas otras facciones. Los subjuegos del - lenguaje han sido desarrollados para enseñar la programación, -- PL/C, para aplicaciones en micro procesadores, PL/M, y para programación de sistemas, PL/S.

El PASCAL es un lenguaje estructurado desarrollado en los 70's como un lenguaje para enseñar técnicas modernas en el desarrollo del Software. El PASCAL es descendiente directo del ALGOL y contiene muchas de las mismas facciones. Ha -- sido implementado en computadoras de todos tamaños y muestra una promesa particular como un lenguaje de desarrollo para aplicaciones en microcomputadora.

El lenguaje de programación "C" fue desarrollado como el lenguaje primario para el sistema operativo UNIX, de la Compañía Bell Laboratories y AT y T, pero también ha sido implementado independientemente del sistema UNIX en una variedad de mini y micro computadoras. El C fue desarrollado para la sofisticada Ingeniería Software y contiene facciones poderosas - que le da una flexibilidad considerable.

El ADA es un lenguaje desarrollado bajo las - auspicias del Departamento de Defensa de los Estados Unidos como un nuevo estandard para sistemas empotrados. El ADA es parecido al PASCAL en su estructura y notación. El ADA soporta un -- juego de facciones de tiempo real que incluyen, manejo de interrupciones, multiprogramación, comunicación interproceso y operaciones a un nivel dependiente de la máquina.

Los lenguajes especializados son caracteriza-

dos por sus formas sintáticas inusuales que han sido diseñadas especialmente para una aplicación. Cientos de lenguajes especializados se utilizan hon en día. En general, tales lenguajes tienen pocos usuarios. El siguiente listado contiene algunos lenguajes representativos.

APL.- Un lenguaje extremadamente consiso y fuerte, diseñado para la manipulación de arreglos y vectores.

APT.- El lenguaje diseñado para herramientas controladas numéricamente.

BLISS.-Diseñado para el desarrollo de compiladores y sistemas operativos.

FORTH.-Diseñado para el desarrollo de microprocesadores.

LISP.- Este lenguaje es para la manipulación de símbolos y procesos de listados encontrados en problemas combinatorios.

SNOBOL.-Diseñado para la manipulación y procesamiento de "strings".

Del punto de vista de la Ingeniería Software, los lenguajes especializados ofrecen ventajas y desventajas, porque éstos lenguajes han sido diseñados para una aplicación.

ESTILO DE CODIFICACION :

Después que el código fuente sea generado, la función del módulo debe de ser aparente sin la referencia de la especificación del diseño. O sea en otras palabras, el código --

debe ser entendible. El estilo de codificación circunda a la filosofía de codificación que enfatiza, simplicidad y claridad.

La documentación interna del código fuente -- inicia con la selección de nombres identificadores (variables ó - rótulos), continúa con los comentarios y concluye con la organización visual del programa. El Software debe de contener documentación interna. Los comentarios proveen al desarrollador con los medios de comunicación con otros lectores del código fuente. Los comentarios pueden proveer una guía clara para el entendimiento durante la última fase del ciclo de vida del Software, mantenimiento.

Hay muchas guías que se han propuesto para el comentario. Los comentarios de piólogo y funcionales son las dos categorías que requieren de diferentes acceso. Los comentarios de piólogo deben aparecer al principio de cada módulo. El formato para tales comentarios es:

1. El estatuto del propósito que indica la función de - cada módulo.
2. Una descripción de interfase que incluye:
   a) Un ejemplo de la secuencia de llamado
   b) Una descripción de todos los argumentos
   c) Una lista de los módulos subordinados
3. Una discusión de datos pertinentes como variables - - importantes y su uso, restricciones y limitaciones.
4. Una historia del desarrollo que incluye:
   a) El autor del diseño del módulo
   b) El auditor revisor y fecha

c) Fechas y descripciones de modificaciones

Los comentarios descriptivos están emportrados en el cuerpo del código fuente y se utilizan para describir las - funciones de procesamiento. Una guía primaria expresada por Van - Tassel: " Los comentarios deben proveer algo extra, no solo para frasear el código". Además, los comentarios descriptivos deben :

* Describir bloques de código, en lugar de comentar - cada línea.
* El uso de líneas en blanco o identaciones, para que las líneas de comentario sean distinguibles del código.
* Sea correcto, comentarios incorrectos o erróneos - son peores que si no tuviera.

Con identificadores propios y buenos comenta - rios, se asegura una documentación interna adecuada.

La construcción del flujo lógico del Software es establecido durante el diseño. La construcción de estatutos - individuales es parte del paso de codificación. La construcción del estatuto debe seguir la regla: Cada estatuto debe ser simple y directo, el código no debe ser convolucionado para afectar la eficiencia.

Los estatutos individuales del código fuente pueden ser simplificados por :

* Evitando el uso de pruebas condicionales complicadas
* Eliminación de pruebas en condiciones negativas
* Evitando sobre bifurcaciones o condiciones.
* Utilizando paréntesis para clasificar las expresiones lógicas ó aritméticas.

Cada una de éstas guías procuran "mantenerlo simple".

EFICIENCIA :

En los sistemas hay una tendencia a utilizar eficientemente los recursos críticos. Los ciclos del procesador y las localidades de memoria primaria son visualizados como recursos críticos y el paso de codificación es el último punto donde se pueden reducir los microsegundos o los bits. Aunque hay tres reglas que debemos conocer :

1. La eficiencia es un requerimiento de ejecución y se debe establecer durante el análisis de requerimientos Software. El Software debe ser tan eficiente como es requerido, no tan eficiente como humanamente posible.

2. La eficiencia es mejorada con el buen diseño.

3. La eficiencia del código y el código van agarrados de la mano.

En general, no sacrificar claridad, legibilidad, ó lo correcto por mejoramientos no esenciales en la eficiencia.

La eficiencia del código fuente está ligado direc-

tamente con los algoritmos definidos durante el diseño detallado. Pero, el estilo de codificación puede tener un efecto en la velocidad de ejecución y requerimiento de memoria. El siguiente juego de guías pueden aplicarse cuando el diseño detallado es traducido al código :

* Simplificar las expresiones aritméticas y lógicas .
* Evaluar cuidadosamente las bifuraciones anidadas -- para determinar si las expresiones o estatutos se - puedan salirse.
* Cuando sea posible, evitar el uso de arreglos multidimensionales.
* Cuando sea posible, evitar el uso de puntadores y - listados complejos.
* Utilize operaciones aritméticas rápidas.
* No mezcle tipos de datos
* Utilize expresiones aritméticas enteras ó booleanas cuando sea posible.

Existen muchos compiladores que tienen facciones optimizadoras que automáticamente generan un código eficiente, -- utilizando las guías anteriores.

Las restricciones de memoria en las macrocomputadoras son una cosa del pasado. La administración de memoria vir -- tual proveen a la aplicación con un enorme espacio lógico direccionable. La eficiencia de la memoria para tales ambientes no pueden- igualarse a la utilización de memoria mínima. En lugar, la eficien

cia de la memoria debe tomarse en cuenta las características de compaginación del sistema operativo.

En general, la localidad del código o el mantenimiento del dominio funcional a través de construcciones estructuradas es un método excelente para reducir la compaginación y - por lo tanto, incrementar la eficiencia.

Hay dos clases Entradas/Salidas (I/O) que se - deben considerar cuando se discute la eficiencia : El I/O dirigida al humano y el I/O dirigida a otro periférico. Del punto de vista del diseño detallado y de la codificación, unas guías que- pueden mejorar la eficiencia del I/O, como sigue:

* Todo el I/O debe ser compactada para reducir la comunicación.
* Para la memoria secundaria (disco), el método de acceso más simple aceptable debe ser seleccionado y utilizado.
* El I/O a memorias secundarias deben ser bloqueadas.
* El I/O a terminales e impresoras deben reconocer - las facciones del equipo que pueda mejorar la calidad ó velocidad.

Hay que recordar que un I/O super eficiente - no sirve sino se puede entender .

C A P I T U L O XI

PRUEBAS Y CONFIABILIDAD DEL SOFTWARE :

Las pruebas del Software es un elemento crítico para asegurar la calidad del Software y representa la última revisión de las especificaciones, diseño y codificación.

La visibilidad aumentada del Software como un elemento del sistema, los costos asociados con una falla del Software son las fuerzas motivadoras para unas pruebas bien planeadas. Es usual para una organización de desarrollo Software gastar un 40% del esfuerzo total del proyecto en pruebas.

En el caso extremo, las pruebas del Software relacionado con humanos (control de vuelos ó monitoreo de un reactor nuclear) los costos pueden ser 3 a 5 veces más de todos los pasos de la Ingeniería Software combinados.

En este capítulo discutiremos tres tópicos interrelacionados. El primero, pruebas del Software, es un paso planeado en el proceso de Ingeniería Software. Como otros pasos, los productos derivados de las pruebas se convierte en parte de la configuración del Software.

Las pruebas nos guían invariablemente al segundo tópico de discusión. *Debugging* ó localización y corrección de errores. Más un arte que una ciencia, el debugging diagnostica los errores del programa y los corrige. Los resultados de las -

pruebas también nos puede guiar a una consideración de confiabilidad, el tercer tópico. Nosotros procuramos garantizar la confiabilidad, mientras al mismo tiempo, desarrollar modelos de predicción de fallas para anticipar problemas. Todavía, debemos confiar en una serie de pasos de pruebas como la única garantía - - práctica de la confiabilidad del Software.

PRUEBAS DEL SOFTWARE :

Las pruebas presentan una anomalía interesante para el Ingeniero Software. Durante los pasos previos de planeación y desarrollo, el Ingeniero intenta construir un Software de un concepto abstracto a una implementación tangible. Ahora -- vienen las pruebas. El Ingeniero crea una serie de casos prueba, que su intención es demolir el Software que ha construído. Las - pruebas es un paso en el proceso de Ingeniería Software que puede ser visto psicológicamente como, destructivo en lugar de constructivo.

En un libro sobre pruebas del Software, Glen Myers menciona un número de reglas que pueden utilizarse como -- objetivos de prueba:

1. Las pruebas es un proceso de ejecución del programa con la intención de encontrar un error.
2. Un buen caso de prueba, es aquel que tiene una alta probabilidad de encontrar un error indescubierto.
3. Una prueba exitosa es aquella que descubre los - - errores indescubiertos.

Los objetivos anteriores implican un cambio dramático en el punto de vista. Se mueven al contrario del punto de - vista común de una prueba exitosa del cuál ningún error se ha -- descubierto. El objetivo principal es de diseñar pruebas que sistemáticamente descubren diferentes tipos de errores.

Hay dos clases de entrada que se proveen para pro - bar el proceso:

1. La configuración del Software que incluye la especificación de requerimientos Software, especificación de diseño y el código fuente.

2. La configuración de prueba que incluye el plan y procedimiento de prueba, casos de prueba, y resultados esperados. La configuración de prueba es un subjuego de la configuración del Software cuando es considerado el ciclo de vida completo.

Las pruebas son consideradas y todos los resultados evaluados. Esto es, los resultados de las pruebas son comparadas con los resultados esperados. Cuando datos erróneos son descubiertos, implica un error y comienza el debugging.El proceso de - debugging es la parte más inpredicable del proceso de pruebas. Un error que tenga una discrepancia de 0.01% entre los resultados encontrados y esperados puede tomar una hora, un día ó un mes para diagnosticar y corregir. Es la incertidumbre inherente en el debugging que hace difícil las pruebas para programar la confiabili - dad.

Como los resultados de las pruebas se coleccionan y se evaluan, una indicación cualitativa de la confiabilidad del Software empieza a notarse. Si errores severos que requieren modificación de diseño son encontrados con regularidad, la calidad y confiabilidad del Software son sospechadas y más pruebas son indicadas. Si en cambio, las funciones Software aparecen trabajando propiamente y se encuentran errores son fáciles de corregir, una de dos conclusiones se pueden tomar. La primera, la calidad y confiabilidad del Software son aceptables. O, la segunda, las pruebas son inadecuadas para descubrir los errores. Finalmente, si las pruebas no descubren errores, hay poca duda que la configuración de pruebas no le fue dada bastante razonamiento y que los errores existen en el Software. Estos errores eventualmente serán descubiertos por el usuario y corregidos por desarrollador durante la fase de mantenimiento, cuando los costos pueden ser 40 veces el costo durante la fase de desarrollo.

Los resultados acumulados durante las pruebas pueden ser evaluadodas en una manera más formal. Los modelos de confiabilidad utilizan datos relacionados con los errores para predecir futuras ocurrencias de errores y de aquí, la confiabilidad.

Cualquier producto puede ser probado en una de dos formas :

1. Si la función que el producto debe ejecutar es conocida, las pruebas pueden ser conducidad para demostrar que cada función es totalmente operacional.

2. Si la función interna del producto son conocidos, se puede conducir las pruebas para asegurarse que las operaciones internas se ejecuten según a la - especificación.

La primera prueba se le denomina pruebas de caja-- negra y a a la segunda, pruebas de caja blanca.

Cuando es considerado el Software, las pruebas de caja negra aluden las pruebas que deben conducirse en las in-- terfases del Software. Esto es, los casos de prueba que la entrada es propiamente aceptada y que la salida es producida - - correctamente y que la integridad de la informaxión externa es mantenida. La prueba de caja negra examina algunos aspectos -- del modelo fundamental del sistema con poca consideración por- la estructura lógica interna del Software.

Las pruebas de caja blanca del Software es predecible en una examinación del detalle procedual. Los caminos lógicos a través del Software son probados por los casos pruebas - que ejercitan juegos específicos de condiciones y/o bifurcaciones.

El estado del programa puede ser examinado en varios puntos para determinar si el estado esperado ó asertado - corresponde al estado actual.

Los atributos de las pruebas de caja negra y blanca pueden ser combinados para proveer un acceso para validar la interfase del Software y asegurar la selectividad de las funcio -

nes internas del Software son correctas.

Las pruebas dentro del contexto de la Ingeniería Software es una serie de cuatro pasos que son implementados secuencialmente . Inicialmente, las pruebas se enfocan en cada módulo individual, asegurándose que funcionen apropiadamente como una unidad, recibiendo el nombre de pruebas de unidad. En seguida, los módulos se deben asemblar ó integrados para formar un paquete de Software completo, pruebas de integración, direcciona las salidas asociadas con los problemas de verificación y asemblaje. Finalmente, los requerimientos de validación establecidos durante la fase de planeación -- deben de probarse. Las pruebas de validación provee la seguridad final que el Software satisfaga todos los requerimientos funcionales y de ejecución.

El último paso esta afuera del límite de la Ingenie - ría Software y en un campo mayor, o sea el de la Ingeniería de Sistemas Computacionales. El Software, una vez validada se debe combinar con otros elementos del sistema, el Hardware e información. Las pruebas del sistema verifica que todos los elementos se engranen -- propiamente y que la función y ejecución en general del sistema son logradas.

PRUEBAS POR UNIDAD :

Las pruebas por unidad se enfoca en el esfuerzo de - verificación en la unidad más pequeña del diseño del Software, el - módulo. Con la descripción del diseño detallado utilizado como guía, los caminos importantes de control son probados para descubrir erro

en los límites del módulo. La complejidad relativa de las pruebas y los errores sin descubrir es limitado por el objetivo - establecido para la prueba por unidad. Las pruebas por unidad son orientadas a las pruebas de caja blanca y el paso se puede conducir en paralelo para varios módulos.

Hay cinco características primarias de un - módulo que son evaluadas durante la prueba por unidad :

* La interfase del módulo
* La estructura local de datos
* Caminos importantes de ejecución
* Caminos de manejo de errores
* Las condiciones limitantes que afectan a los anteriores.

Se requieren pruebas del flujo de datos a - - través de la interfase del módulo antes que otra prueba sea iniciada. Si los datos no entran y existen propiamente, todas las - demás pruebas fallan. La estructura local de datos para un módulo es una fuente común de errores. Se deben diseñar casos de - - prueba para descubrir errores, en las siguientes categorías :

1. Declaración impropia e inconsistente
2. Iniciación errónea
3. Nombres de variables incorrectos
4. Inconsistencia de tipos de datos
5. Excepciones de bajo flujo, sobre flujo y direccionamiento.

Además de estructuras locales de datos, el impacto de los datos globales en el módulo se debe investigar si es posible durante la prueba por unidad.

La prueba selectiva de caminos de ejecución es una tarea esencial durante la prueba por unidad. Los casos prueba se deben designar a descubrir errores debidos a computación errónea, comparaciones incorrectas ó flujo de control incorrecto. Entre los errores más comúnes en la computación son : precedencia aritmética incorrecta o mal entendida, operaciones en modo mixto, inicialización incorrecta, precisión sin exactitud, representación simbólica incorrecta de una expresión.

La prueba de límite es la última y probablemente la más importante tarea de la prueba por unidad. El Software a veces falla en los límites. Esto es, los errores a veces ocurren cuando el enésimo elemento de un arreglo no dimensional es procesado, cuando la iésima repetición de la hiforcación se encuentra. Los casos prueba que ejercitan la estructura de datos, flujo del control y valores de datos justamente, abajo, on y arriba de la máxima y mínima es común descubrir errores.

PRUEBAS DE INTEGRACION :

Las pruebas de integración es una técnica sistemática para asemblar el Software mientras que al mismo tiempo conduciendo pruebas para descubrir errores asociados con las interfases el objetivo es tomar los módulos probados por unidad y construír una estructura Software que ha sido dictado por el diseño.

Integración de arriba hacia abajo es un método incremental del asemblaje de la estructura del Software. Los módulos son integrados moviéndose hacia abajo a través de la jerarquía de control, iniciando con el módulo de control principal. Los módulos subordinados al módulo principal de control son incorporados a la estructura en una forma de profundidad primero.

El proceso de integración es ejecutado en una serie de cinco pasos :

1. El módulo de control principal es utilizado como la prueba impulsora y los talones son substituídos por todos los módulos subordinados al módulo de control principal.

2. Dependiendo del método de integración seleccionado los talones subordinados son reemplazados uno a la vez por los módulos.

3. Las pruebas son conducidas como cada módulo es integrado.

4. Al término de cada juego de pruebas, otro talón es reemplazado con otro módulo.

5. La prueba de regresión puede ser conducida para asegurarse que no se han introducido nuevos errores.

La estrategia de la integración arriba hacia abajo verifica los puntos de control ó desición temprano en el proceso de pruebas.

Las pruebas de integración de abajo hacia arriba, -

como su nombre lo implica, inicia el asemblaje y pruebas con los módulos en los niveles más bajos de la estructura del Software. Porque los módulos son integrados de abajo hacia arriba, el procesamiento requerido para los módulos subordinados a un nivel -- dado siempre está disponible y la necesidad de los talones se - elimina.

La estrategia de la integración de abajo hacia arriba puede ser implementada siguiendo éstos pasos:

1. Los módulos en los niveles bajos son combinados en grupos que ejecutan una subfunción específica.
2. Un programa de control para pruebas es escrito para coordinar los casos prueba de entrada y salida.
3. El grupo es probado.
4. El programa de control de pruebas son removidos y los grupos combinados, moviéndose hacia arriba de la estructura Software.

Como la integración se mueve hacia arriba, la necesidad de utilizar programas de control de pruebas se aminora, si los dos primeros niveles de la estructura Software son integrados de arriba hacia abajo, el número de programas control de pruebas - se pueden reducir substancialmente y la integración de los grupos se simplificarán.

Los siguientes criterios y pruebas correspondientes son aplicados a todas las fases de prueba.

Integridad de la Interfase.- Las interfases internas y externas son probadas como cada módulo se incorpora en la estructura.

Validación Funcional.- Las pruebas designadas a descubrir errores funcionales son conducidas.

Contenido de Información.- Las pruebas designadas a descubrir errores asociados con estructura de datos locales o globales son conducidas.

Ejecución.- Las pruebas diseñadas para verificar los límites de ejecución establecidos durante el diseño del Software son conducidos.

PRUEBAS DE VALIDACION :

En la culminación de las pruebas de integración, el Software es asemblado completamente como un paquete, -- errores de interfase han sido descubiertos y corregidos, y una serie final de pruebas de Software, la prueba de validación puede iniciar. La validación puede ser definido en muchas formas, - pero una definición simple, la validación es exitosa cuando las funciones Software dan los resultados esperados por el usuario.

La validación del Software es lograda a través de una serie de pruebas de caja negra que demuestran la conformidad con los requerimientos. Después de cada prueba de validación ha sido conducida, una de dos condiciones posibles existen la primera, las características de función ó ejecución se - conforman a las especificaciones y son aceptadas. La segunda, -

una desviación de las características es descubierta y una lista de deficiencia es creada. Una desviación o error descubierto en este estado del proyecto raramente puede ser corregido antes de terminar la programación. Es, a veces, necesario negociar con el usuario para establecer un método para resolver las deficiencias.

PRUEBAS DEL SISTEMA :

Al inicio de este libro se dijo que el Soft - ware es solo un elemento de un sistema computacional. El Softwa- re es incorporado con otros sistemas (Hardware e información) y una serie de pruebas de integración y validación son conducidas. Los pasos tomados durante el diseño y pruebas del Software pue - den mejorar la probabilidad del éxito de la integración del - - Software en un sistema grande.

El último paso en la prueba del sistema es - llamada la prueba de aceptación. Conducida por el usuario en lu- gar del desarrollador del sistema, la prueba de aceptación pue- de tener un rango desde una prueba informal hasta una serie de pruebas planeadas y sistemáticamente ejecutadas. La prueba de - aceptación puede ser conducida en un período de semanas ó meses, descubriendo errores acumulativos que pueden degradar el sistema sobre el tiempo.

EL ARTE DE DEBUGGING :

Las pruebas del Software es un proceso que - - puede ser planeado y especificado sistemáticamente. Hemos visto -

que la meta de las pruebas es de descubrir errores. El debugging ocurre como consecuencia de una prueba exitosa.

El Ingeniero de Software, evaluando los resultados de la prueba, es a veces confrontado con una indicación simptomática del problema del Software. Esto es, una manifestación externa del error y la causa interna del error que pueda no tener una relación obvia con otra. La pobremente entendida proceso mental que conecta el síntoma a una causa del problema Software es el debugging.

Indiferentemente del acceso que se toma, el debugging tiene un objetivo, encontrar y corregir la causa del error. El objetivo es realizado por una combinación sistemática de evaluación, intuición y suerte. En general, se pueden proponer tres categorías para el debugging :

* Fuerza Bruta
* Eliminación de la causa
* Retro traceando

La categoría de fuerza bruta de debugging es probablemente la más común y el método más ineficiente para aislar la causa del error. Si una filosofía de "deja que la computadora encuentre el error" es utilizada, se toma mucha memoria, se invocan trazos de tiempo de ejecución y el programa está cargado con puros estatutos WRITE. Pero la masa de información producida puede guiarnos al éxito, y frecuentemente nos guía al gasto inútil de esfuerzo y tiempo.

El segundo acceso al debugging es eliminación de la causa es manifestado por inducción o deducción. Los datos relacionados a las ocurrencias de error son organizadas para aislar las causas potenciales. Una hipótesis de la causa es derivada y los datos anteriores son utilizados para aprobar o reprobar la hipótesis. Alternativamente, una lista de todas las causas probables es desarrollada y las pruebas son conducidas para eliminar o substanciar cada causa. Si las pruebas iniciales indican que una hipótesis de la causa particular muestra algo, los datos son refinados para tratar de aislar el error.

La retrotraceabilidad es una técnica del debugging común que puede utilizarse exitosamente en programas pequeños. Inciando donde se localiza el simptoma, el código fuente se retrotracea manualmente hasta el sitio de la causa es encontrado. Desafortunadamente, el número de líneas fuente se incremente y el número de caminos potenciales de retrotraceabilidad se convierten en muchos.

Cada una de las técnicas descritas pueden suplementarse con herramientas de debugging. Podemos aplicar una gran variedad de compiladores de debugging, auxilios dinámicos de debugging (traceadores), y generadores automáticos de casos de prueba, pero, las herramientas no son buenos substitutos para una evaluación cuidadosa basado en el documento completo del diseño del Software y un código fuente claro.

CONFIABILIDAD DEL SOFTWARE :

La confiabilidad de cualquier sistema técnico es - medido en términos estocasticos. Esto es, la confiabilidad es la probabilidad que el sistema se ejercite su función asignada bajo condiciones ambientales específicas para un período de tiempo - dado. Si consideramos un sistema computacional, la confiabilidad del Hardware se puede medir como el tiempo medio entre fallas - - (MTBF) donde

$$MTBF = MTTF + MTR$$

Los términos MTTF y MTR son el tiempo medio para - la falla y el tiempo medio para reparar la falla, respectivamente fallas del Hardware son generalmente causadas por deterioración - por el tiempo. Entonces, la confiabilidad del Hardware puede ser determinada utilizando una combinación de datos empíricos y modelos estocásticos.

La confiabilidad del Software puede ser caracterizado en términos que se acerca a la definición de confiabilidad para sistemas técnicos. Goodenoogh define la confiabilidad del Software como, "La frecuencia de fallas y lo crítico de las fallas del programa cuando las fallas son en efecto inaceptable o un comportamiento bajo condiciones de operación permisibles". Como el Hardware, la confiabilidad del Software puede ser representada por la relación en que los errores son encontrados y descubiertos. A diferencia del Hardware, hay menos evidencia que los datos de errores empíricos que se puedan utilizar para desarrollar modelos precisos de predicción para la confiabilidad del Software. Un intento para desarrollar una teoría matemática de la confiabilidad del

Software ha resultado en un número de modelos prometedores.

Los modelos de confiabilidad del Software - normalmente caen en una de las siguientes categorías :

* Modelos derivados de la teoría de confiabilidad - Hardware.
* Modelos basados en las características internas - del programa.
* Modelos desarrollados plantando en el Software - con errores conocidos y evaluando el número de -- errores con semilla detectados contra los errores actuales detectados.

Las asunciones hechas por los modelos en la primer categoría son presentados por Sukert y Boel: El tiempo de debugging entre ocurrencias de error tiene una distribución exponencial con la relación de ocurrencias de error que es -- proporcional al número de errores restantes. Cada error descubierto es inmediatamente removido, decrementando el número total de errores por 1 y la relación de fallas entre errores es constante. La validez de cada asunción puede ser cuestionada.

La segunda categoría computa un número predecible de errores que existen en el Software. Los modelos basados en una relación cuantitativa se derivan como una función - de la medida de complejidad del Software, una relación específica del diseño o atributos orientados al código del programa para estimar el número inicial de errores que se puedan esperar en un programa dado.

Modelos con semillas puede ser utilizado como una indicación de la confiabilidad del Software, o más prácticamente, - una medida del "poder descubrir errores" de un juego de casos - prueba. Un programa se le implantan semillas al azar con un humano conocido de errores de calibración.

El programa se corre, y la probabilidad de encontrar -- "J" errores reales de una población total de "J" errores pueden ser relacionados a la probabilidad de encontrar "K" errores con semilla de "K" errores de calibración ue son implantados en el código .

C A P I T U L O XII

EL MANTENIMIENTO DEL SOFTWARE :

El mantenimiento ha sido caracterizado como un Lurte. Esperando que lo que sea inmediatamente visible sea todo lo que es. En realidad, se sabe que una enorme masa de problemas potenciales y los costos están debajo de la superficie. El mantenimiento del existente Software se estima sobre el 60% de todo el esfuerzo gastado en la organización de desarrollo. El porcentaje continúa a incrementar como más Software es desarrollado. En el horizonte se puede ver la limitación del mantenimiento de la organización de desarrollo Software que no pueda producir nuevo Software porque está gastando todos los recursos disponibles en mantener el Software viejo.

Un lector no iniciado puede preguntar porque se requiere mucho mantenimiento y porque se gasta tanto esfuerzo. Rochkind provee una contestación parcial:

"Los programas de computadora siempre están cambiando. Hay errores que arreglar, engrandecimientos que agregar, y optimizaciones que hacer. No hay solo la versión de este año para cambiar, pero también la versión del año pasado y la versión del año que entra. Aparte a los problemas que sus soluciones requieren los cambios en primer lugar, y los cambios mismos crean problemas adicionales".

A través de este libro se ha discutido una metodología de Ingeniería Software. Su meta primaria es reducir el esfuerzo gastado en el mantenimiento. En este capítulo, la fase final del ciclo de vida del Software, el mantenimiento Software.

UNA DEFINICION DEL MANTENIMIENTO DEL SOFTWARE :

Podemos definir el mantenimiento describiendo cuatro actividades que se emprenden después que el programa sea liberado para su utilización.

La primera actividad de mantenimiento ocurre porque no es razonable asumir que las pruebas del Software descubrirán errores latentes en un sistema grande. Durante la utilización de cualquier gran sistema, errores ocurrirán y serán reportados al desarrollador. El proceso que incluye el diagnóstico y corrección de uno o más errores se llama mantenimiento correctivo.

La segunda actividad que contribuye a la definición del mantenimiento ocurre por el cambio rápido que se encuentra en cada aspecto de la computación. Nuevas generaciones de Hardware se anuncian en ciclos de 36 meses, nuevos sistemas aparecen regularmente, equipo periférico y otros elementos del sistema son frecuentemente modificados.

La vida útil de la aplicación del Software puede sobre pasar los diez años, sobreviviendo el ambiente del sistema por el cuál fue originalmente desarrollado. Entonces, el mantenimiento adaptivo es la actividad que modifica el Software para que propiamente sea interfase del ambiente cambiante, es común --

y necesario.

La tercera actividad que puede ser aplicada a la definición del mantenimiento ocurre cuando el paquete Soft - ware es exitoso. Como se utiliza el Software, se recomiendan - nuevas capacidades, modificaciones a funciones existentes y engrandecimientos son recibidos por los usuarios. Para satisfacer las peticiones en esta categoría. Se ejecuta el mantenimiento - perfectivo. Esta actividad estima la mayoría del esfuerzo gasta do en el mantenimiento del Software.

La cuarta actividad del mantenimiento ocurre- cuando el Software es cambiado para mejorar el mantenimiento fu turo ó confiabilidad o para proveer una mejor base para futuros engrandecimientos ó mejoramientos. A veces llamada mantenimiento preventivo, ésta actividad es todavía rolativamente rara en el mundo del Software.

CARACTERISTICAS DEL MANTENIMIENTO :

El mantenimiento Software, hasta recientemente ha sido la fase negada en el ciclo de vida del Software. La literatura sobre mantenimiento contiene pocas entradas cuando - es comparado con las fases de planeación y desarrollo. Poca investigación o datos de producción han sido recolectados sobre - este tema y pocos métodos técnicos han sido propuestos.

Para entender las características del manteni miento del Software, consideremos el tópico de tres puntos de - vista:

1. Las actividades requeridas para efectuar la fase de mantenimiento y el impacto del acceso de la Ingeniería de-Software (a la falta de) sobre la eficiencia de tales actividades.

2. Los costos asociados con la fase de mantenimiento.

3. Los problemas que frecuentemente se encuentran cuando - se efectúa el mantenimiento.

El costo del mantenimiento ha incrementado- durante los pasados 20 años. Aunque los promedios industriales - son difíciles de decir y están abiertos a una interpretación amplia, la organización típica de desarrollo Software gasta entre- el 40 y 60% de las entradas en mantenimiento.

El costo del mantenimiento es muy obvio. -- Pero, costos menos tangibles puede ser la causa para mayor preocupación. Un costo intangible del mantenimiento del Software es la oportunidad que se ha pospuesto o perdido porque los recur - sos disponibles se deben canalizar a las tareas de mantenimiento. Otros costos intangibles incluyen:

* Insatisfacción del cliente cuando los pedidos legítimos para reparación ó modificación no pueden ser direccionados a tiempo.

* La reducción de la calidad general del Software como un resultado de los cambios que introducer errores -

latentes en el mantenimiento del Software.

* Solventamiento causado durante los esfuerzos de desarrollo cuando el staff deben de trabajar en tareas de mantenimiento.

El costo final del mantenimiento del Software es un decremento dramático en la productividad (medido en líneas de código (LOC)/persona-mes ó $/LOC) que se encuentran cuando es -- iniciado el mantenimiento en programas viejos. Las reducciones - de productividad de 40 a 1 han sido reportadas. Esto es, un es - fuerzo de desarrollo que costó $25.00 por LOC para desarrollar, puede costar $1000.00 por LOC de mantención.

El esfuerzo gastado en mantenimiento puede ser - dividido en actividades productivas (análisis, evaluación, diseño de modificaciones y codificación) y otras actividades como - tratar de entender que hace el código, tratar de interpretar las estructuras de datos, características de las interfases y límites de ejecución. La siguiente expresión provee un modelo para - el esfuerzo de mantenimiento.

$$M = p + K \exp(c-d)$$

donde M es el esfuerzo total gastado en mantenimiento, p es el - esfuerzo de productividad, K es una constante empírica, c es la medida de complejidad que se puede atribuír a la falta de estructuración del diseño y documentación, y d es la medida del grado de familiaridad con el Software.

El modelo indica que el esfuerzo y el costo pueden incrementar exponencialmente, si la metodología de desarrollo del Software es pobre (falta de Ingeniería Software) fue utilizada, y la persona o grupo que utilizó la metodología no está disponible para la ejecución del mantenimiento.

MANTENIBILIDAD :

Las características descritas anteriormente son afectadas por la mantenibilidad del Software. La mantenibilidad puede ser definido cualitativamente como la facilidad, con la cuál, el Software es entendido, corregido, adaptado y/o mejorado. Como se ha enfatizado a través de este libro, la mantenibilidad es la meta primaria que guía los pasos de la metodología de la Ingeniería Software.

La mantenibilidad del Software es afectada por muchos factores. Descuido inadvertido en el diseño, codificación y pruebas tienen un impacto negativo en la habilidad de mantener el Software puede tener un impacto negativo similar, aún cuando los pasos técnicos han sido conducidos con cuidado.

Además de los factores que se pueden asociar con la metodología del desarrollo, Kopetz define un número de factores que están relacionados al ambiente de desarrollo:

* La disponibilidad de un staff calificado
* La estructura del sistema entendible
* La facilidad del manejo del sistema

* La utilización de lenguajes de programación estandarizado.
* La utilización de sistemas operativos estandarizados
* La estandarización de la estructura de documentación
* Disponibilidad de casos prueba
* Facilidades del debugging
* Disponibilidad de la computadora para conducir el -- mantenimiento.

Posiblemente, el facor más importante que - afecta a la mantenibilidad es planear para la mantenibilidad. Si el Software es visto como un elemento del sistema que inevitablemente sufrirá cambios, los cambios que el Software mantenible que se producirán se incrementarán substancialmente.

La mantenibilidad del Software, como la calidad ó confiabilidad, es un término difícil de medir. Podemos --- amillar indirectamente la mantenibilidad considerando los atributos de la actividad de mantenimiento que podemos medir. Gilb provee un número de medidas de mantenibilidad que se relacionan con el esfuerzo gastado durante el mantenimiento :

1. Tiempo de reconocimiento del problema
2. Tiempo de retardo administrativo
3. Tiempo de colección de herramientas de mantenimiento
4. Tiempo de análisis del problema
5. Tiempo de cambio de especificación
6. Tiempo activo de corrección ó modificación
7. Tiempo de pruebas locales

8. Tiempo de pruebas globales
9. Tiempo de revisión del mantenimiento
10. Tiempo total de recuperación

Cada una de éstas medidas pueden ser grabadas -- sin gran dificultad. Tales datos pueden proveer al administrador con una indicación de la eficiencia de técnicas y herramientas -- nuevas.

En cada nivel del proceso de revisión, la mante - nibilidad se ha considerado. Durante la revisión de requerimien - tos (Capítulo IV), se notaron las áreas de mejoramiento futuro y revisiones potenciales, los puntos de portabilidad del Software - se discutieron, y se consideraron las interfases del sistema que puedan tener un impacto en el mantenimiento. Durante las revisio- nes informales y formales (Capítulo V), la estructura y procedi - miento son evaluadas para la facilidad de modificación, modulari- dad e independencia funcional. Las revisiones del código (Capítulo X) enfatizan el estilo y la documentación interno, dos factores -- que tienen influencia con la mantenibilidad. Finalmente, cada paso de prueba (Capítulo XI) puede proveer sugerencias de partes del -- programa que pueda requerir mantenimiento preventivo antes de que sea formalmente liberado.

Las revisiones de mantenibilidad son conducidas - repetidamente como cada paso en el proceso de Ingeniería Software es completada. La revisión de mantenimiento más formal ocurre en - la conclusión de las pruebas y es llamada la revisión de configu- ración. Discutido en el Capítulo XI, la revisión de la configura - ción asegura que todos los elementos de la configuración del Soft-

ware estén completos, entendibles y archivados para un control de modificación.

B I B L I O G R A F I A

CAPITULO I.

A. OSBORNE, RUNNING WILD, THE NEXT INDUSTRIAL REVOLUTION.
Mc GRAW - HILL

C. BELL, J. MUDGE Y J. Mc NAMARA, COMPUTER INGINEERING.
DIGITAL PRESS, DIGITAL EQUIPMENT Co.

T. BARTEE, DIGITAL COMPUTER FUNDAMENTALS
Mc GRAW - HILL.

CAPITULO II.

J.C. WETHERBE, SYSTEMS ANALISIS FOR COMPUTER BASED INFORMATION SYSTEMS.
WEST PUBLISHING.

T.K. ORR, STRUCTURED SYSTEMS DEVELOPMENT
YOURDON PRESS.

T. DE MARCO, STRUCTURED ANALYSIS AND SYSTEM ESPECIFICATION.
WILEY - INTERSCIENCE.

R. WENING - SYSTEMS ANALYSIS CHECKLIST
AVERBACH PUBLICHERS.

J. KING AND E. SCHREMS, COST BENEFIT ANALYSIS IN INFORMATION SISTEMS DEVELOPMENT.

CAPITULO III.

V. BASILI Y M. ZELKOWITZ, ANALYZING MEDIUM SCALE SOFTWARE DEVELOPMENT
IEEE

C. WALSTON Y C. FELIX, A. METHOD OF PROGRAMMING MEASURMENTS AND ESTIMATION.

IBM SYSTEMS JOURNAL, VOLIB.
V. BASILI, MODELS AND METRICS FOR SOFTWARE.

MANAGMENT AND ENGINEERING
L. PUTMAN, A GENERAL EMPIRICAL SOLUTION TO THE

MACRO SOFTWARE SIZING AND ESTIMATING PROYECT
IEEE

P. NORDEN, USETUL TOOLSFOR PROYECT MANAGMENT
IEEE

R. ESTERLING, SOFTWARE MAN POWER COSTS: A MODEL
DATAMATION.

L. PUTMAN SOFTWARE COST ESTIMATING AND LIFE CYCLE CONTROL.

IEEE COMPUTER SOCIETY PRESS.

CAPITULO V.

E.S. TAYLOR, AN INTERIM REPORT ON ENGINEERING DESIGN
MASSACHUSETTS INSTITUTE AL TECHNOLOGY.

J. DENNIS, MODULARITY
SPRINGER - VERLAG

E. YOURDAN Y L. CONSTANTINE, STRUCTURED DESIGN
PRENTICE - HALL.

CAPITULO VI.

A. WASSERMAN, INFORMATION SYSTEM DESIGN METHODOLOGY
IEEE COMPUTER SOCIETY PRESS.

D. ROSS, J. GOODNEOUGH Y C. IRVINE, SOFTWARE ENGINEERING : PROCESS, PRINCIPLES AND GOALS
IEEE COMPUTER SOCIETY PRESS.

B. BOCHM, CHARACTERISTICS OF SOFTWARE QUIALITY
NORTH HOLLAND PUBLISHING Co.

T. McCALL, FACTORSIM SOFTWARE QUIALITY
GENERAL ELECTRIC Co.

I. Mc CABE, A SOFTWARE COMPEXITY MEASURE
IEEE COMPUTER SOCIETY PRESS

M. HALSTEAD, ELEMENTS AL SOFTWARE SCIENCE
NORTH HOLLAND PUBLISHING Co.

CAPITULO VII.

P. FREEMAN, THE CONTEX OF DESIGN
IEEE COMPUTER SOCIETY PRESS
O. DAHL, E. DIJKSTRA Y C. HOARE, STRUCTURED PROGRAMMING.
ACADEMIC PRESS.

CAPITULO VIII.

J. P. TIEMBLAY Y P.G. SORENSON, AN INTRODUCCION TO
DATA STRUCTURES WITH APPLICATIONS
Mc GRAW - HILL.

E. HOROWITZ Y S. SAHNI, FUNDAMENTAL SOF COMPUTER
ALGORITHMS.

COMPUTER SCIENCE PRESS
J.D. WARNIER LOGICAL CONSTRUCTION OF PROGRAMS
VAN NOSTRAND.

A. WASSERMAN, PRINCIPLES OF SYSTEMATIC DATA
DESIGN AND IMPLEMENTATION.

CAPITULO IX.

E. DIJKSTRA, PROGRAMMING CONSIDERED ASA HUMAN ACTIVITY
NORTH HOLLAND PUBLISHING Co.

N.CHAPIN, A NEW FORMAT FOR FLOW CHARTS
PRENTICE - HALL.

CAPITULO X

T. PRATT, PROGRAMMING LANGUAGES: DESIGN AND IMPLEMENTATION

PRENTICE - HALL

B. KERNIGHAMY B. PLAUGER, THE ELEMENTS OF PROGRAMMING STYLE.

Mc GRAW - HILL.

D. VAN TASSEL, PROGRAM STYLE, DESIGN, EFFICIENCY DEBUGGING AND TESTIMG
PRENTICE - HALL

CAPITULO XI

M. DEUTSCH, VERIFICACION Y VALIDATION
PRENTICE - HALL

W. E. HOWDEN THEORETICAL AND EMPIRICAL STUDIES OF PROGRAM TESTING.
IEEE COMPUTER SOCIETY PRESS.

J.BROWN Y M. LIPOW, TESTING FOR SOFTWARE REALIABILITY.
WILEY

A. BROWN Y W. SAMPSON, PROGRAMMING DEBUGGING
AMERICAN ELSEVIER.

E. MILLER, TUTORIAL : AUTOMATED TOOLS FOR SOFTWARE INGINEERING.
IEEE COMPUTER SOCIETY PRESS.

CAPITULO XII

B. LEINTZ Y E. SWANSON, SOFTWARE MAINTENANIC MANAGMENT.
ADDISON - WILEY

D. FREEDMAN Y B. WEIN BERG. TECHNIQUES OF PROGRAM AND SYSTEM MAINTENANCE
WINTHROP PUBLISHERS.