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, con-- sultante, 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 organi- zación de desarrollo. El administrador del proyecto puede seguir co mo coordinador para facilitar los caminos de comunicación. La meta del analista es el reconocimiento de los elementos básicos del pro blema, 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 sin tetizar 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 especi ficaciones de los requerimientos del software completamente. Este - es un raro caso en el mundo real. Las especificaciones son desarro- lladas 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-

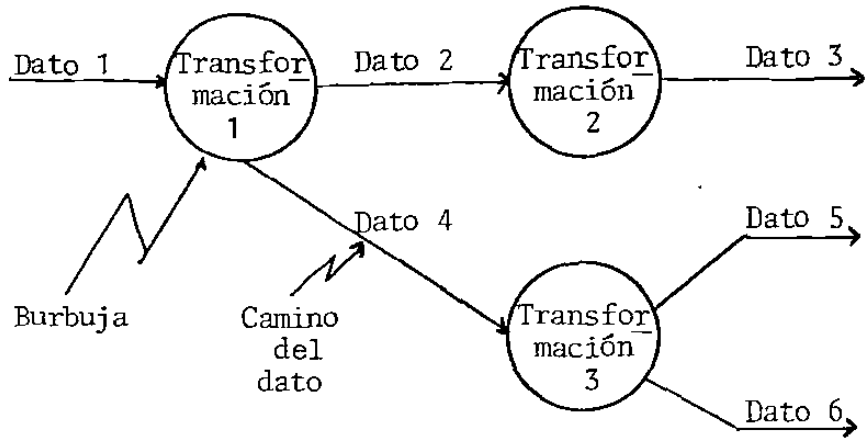
ware exitosa. Estos criterios sirven como base para pruebas durante el desarrollo software. Son escritos formalmente las requerimientos de la especificación para definir las características y atributos - del software. Además se escribe un manual del usuario preliminar.

Se puede parecer raro que el manual del usuario se desarrolle 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 siseño propuestos por Yourdon, Constantine y DeMarco. También es conocido 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 general, el modelo fundamental es refinido en una serie de burbujas, representando cada una, una transformación que ocurre a travéz 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 hibrido, puede ser representado.

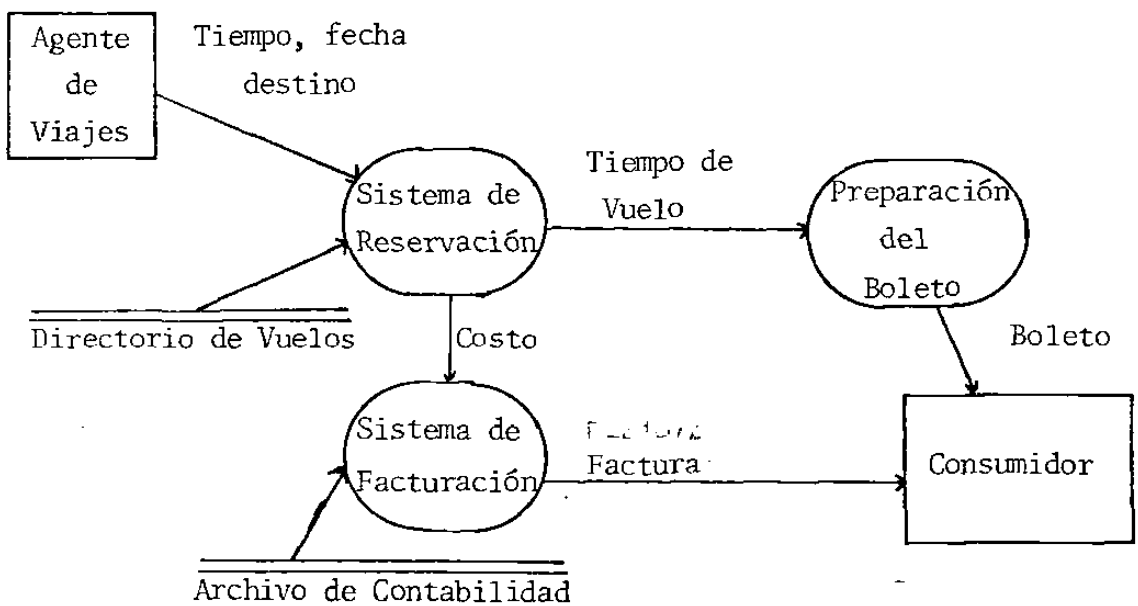
- Cada burbuja puede requerir algo de refinamiento para establecer --

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.

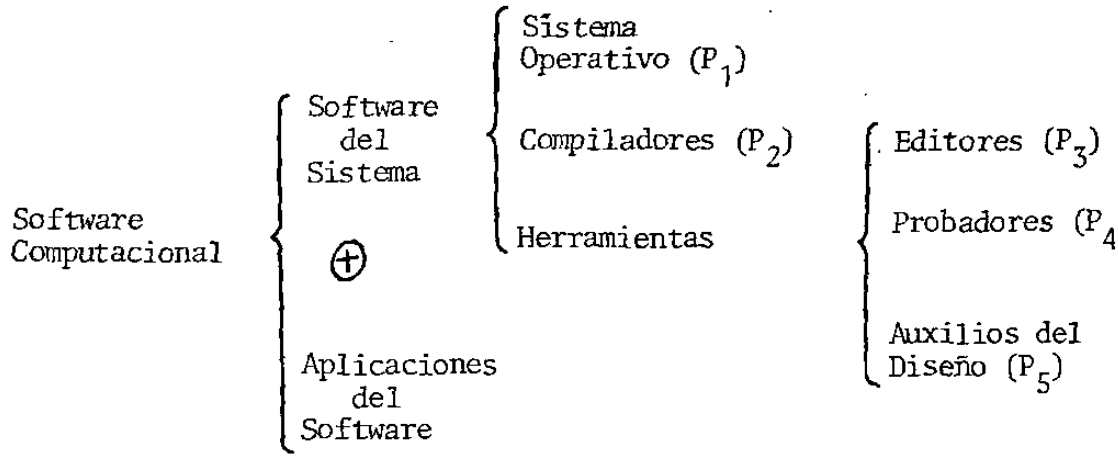
DIAGRAMAS WARNIER

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 dia-- grama 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 catego - rías de información o cantidades. El símbolo de OR exclusivo ($\oplus$)- 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 en trada de un producto Software consiste de información del sistema - Software y de aplicaciones Software. Donde existen P_1 sistemas oper- tivos, P_2 compiladores y una subcategoría de herramientas Software. Bajo esta subcategoría contienen P_3 editores, P_4 probadores y P_5 aux liares 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 interdase al elemento software y resulta en un juego completo de requerimientos 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 determinar 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 administración 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 análisis 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 con-suficiente 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 princiíos 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.

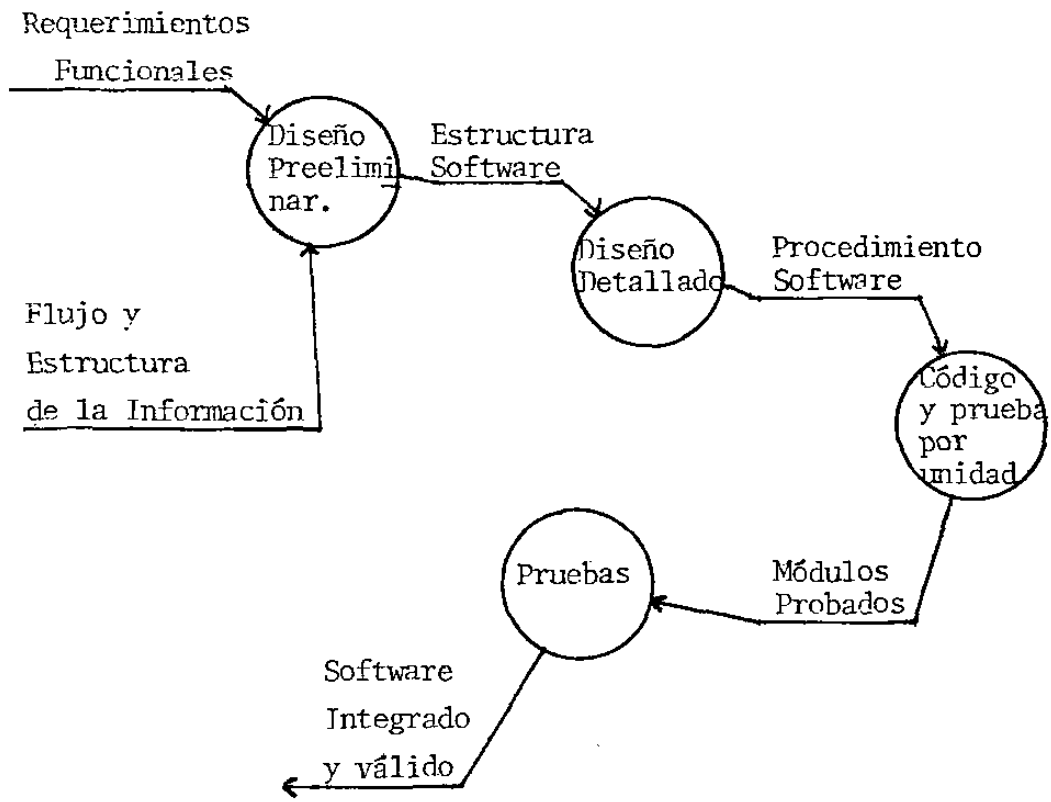
LA FASE DE DESARROLLO :

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 mantenimiento del Software.

EL PROCESO DE DISEÑO :

El Diseño del Software es un proceso a través del cuál los requerimientos son traducidos en una representación del Software. Inicialmente la representación muestra un punto de vista holístico 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 jerárquica que hace uso inteligente del control entre 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 de las 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 los elementos 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 cruzadas 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 diseño 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 documentación de prueba son contenidas en la sección 7.0 del documento 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 procedimiento de prueba no sea completada hasta después en la fase de desarrollo, 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 :

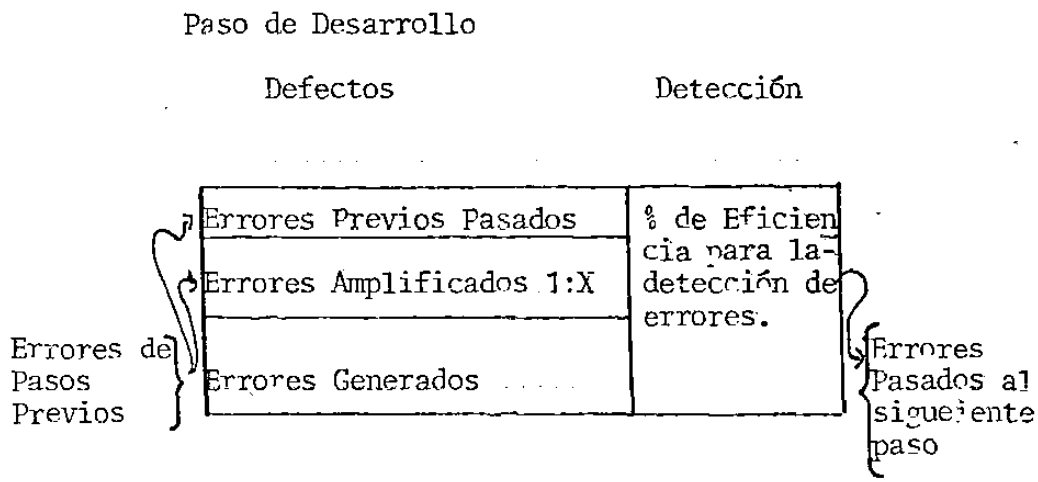
El flujo del proceso de diseño del Software son -- precisados con las revisiones. Una revisión bien planeada es - tan importante para el diseño del Software como los métodos -- técnicos de diseño. Dos filosofías de revisión de diseño coexisten y son aplicadas en diferentes puntos del proceso del diseño. 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 varias 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 Software?

¿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 desemminada 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 implementa - dor del código (programador), quién es responsable de la traducción del diseño a un código fuente, y el probador, es el indi - viduo 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, preparan 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 descu - biertos 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.

CAPITULO 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 refiere 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 resulta en pasar el control a otro módulo. Los mecanismos de activación 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 convencionales tienen una entrada y una salida y son ejecutadas secuencialmente 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 inserta ción 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 coincidental. Un módulo que ejecuta las tareas que lógicamente se relacionan 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 temporal.

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, está 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 entre módulos resulta en un Software más fácil de entender y menos probable a que tenga un efecto de rizo causado cuando los errores 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.

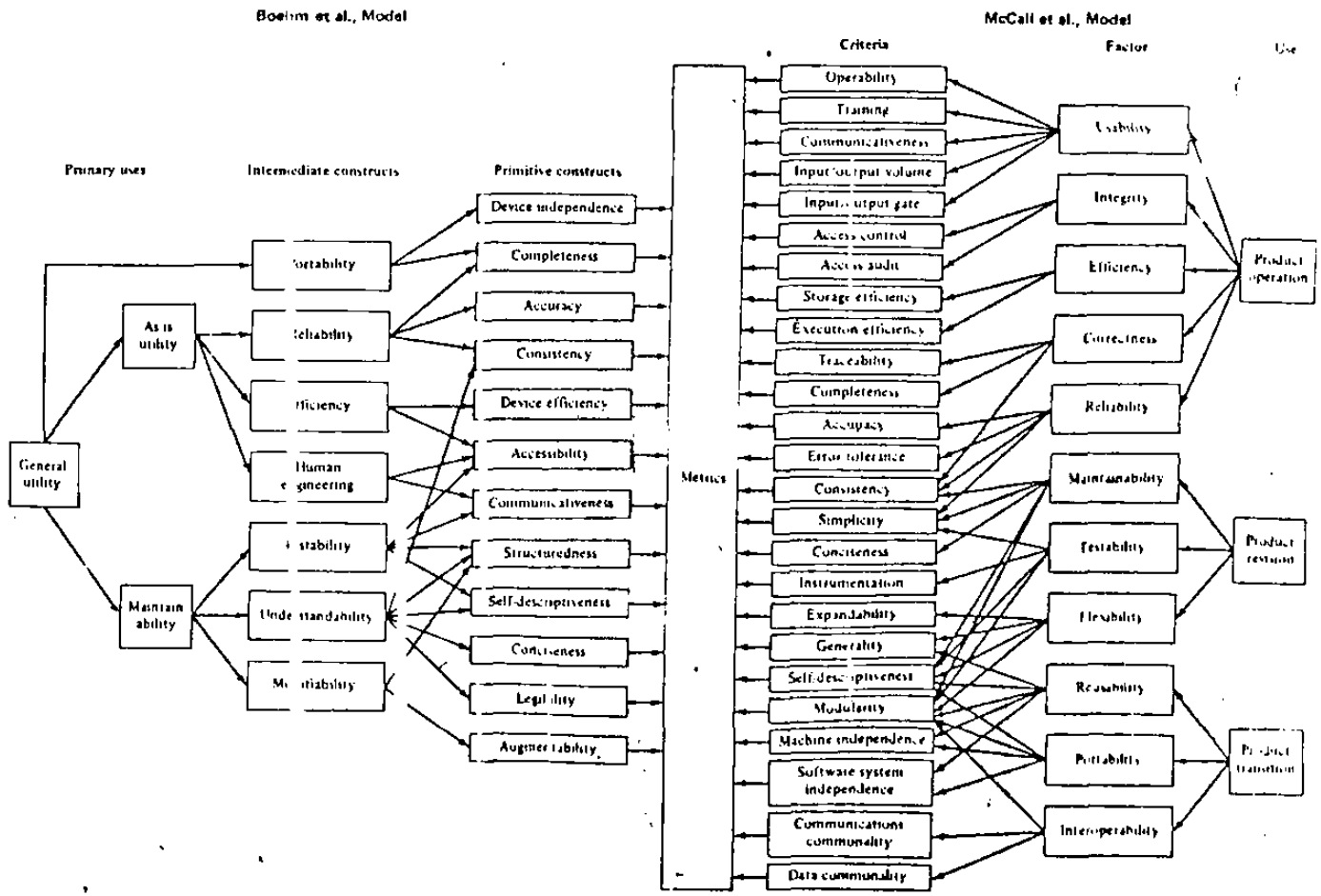
Los modos de acoplamiento anteriormente discutidos ocurren porque las decisiones del diseño hechas cuando -- la estructura fue desarrollada. Las variantes de acoplamiento - externa se pueden introducir durante la codificación.

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.)

3. Retro alimentación para el Ingeniero Software.

Herramientas de información para la administra - ción 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, aplica - ción 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 - cas 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_1 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, l, 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 :

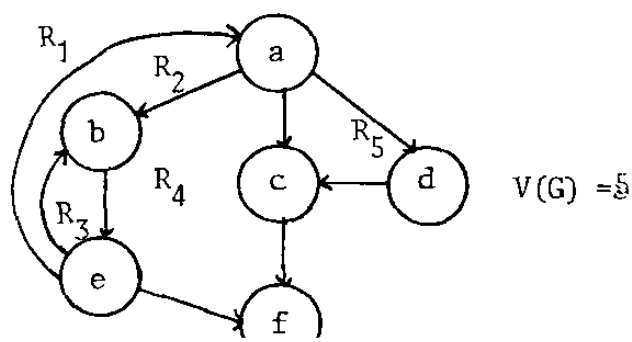
Lenguaje	l Promedio
Inglés (prosa)	2.16
PL/1	1.53
ALGOL/68	2.12
Fortran	1.14
Asemblador	0.88

Aparenta que el nivel de lenguaje implica un nivel de abstracción en la especificación del procedimiento. Lenguajes 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).



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 heurísticas 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 descripción 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 hacer decisiones mayores, a veces de naturaleza estructural. Comparte 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 mi - croprocesador, 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 factoriza - ción.
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 subflujos 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 control 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 de 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 caracte-- rísticas 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 lenguaje máquina para mejorar la eficiencia.

Estos criterios dictan la siguiente frase :

"Hazlo trabajar y luego hazlo más rápido"