UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO
INGENIERÍA EN COMPUTACIÓN

TAREA 8

DISEÑO DE SISTEMAS

QUE PRESENTA:

DAVID MARTÍNEZ PEÑA

ICO 9

NOVENO SEMESTRE

ATLACOMULCO, MÉXICO A 24 DE NOVIEMBRE DE 2008.

DISEÑADO DE SISTEMAS

Qué es diseño

Diseño es el proceso creativo de transformación del problema en una solución. En el diseño de software se utiliza la especificación de requerimientos para definir el problema.

La naturaleza de la solución puede cambiar al describir la solución o al implementarla. La descripción de un sistema puede cambiar durante el ciclo de desarrollo. Muchas veces el cliente, de acuerdo con los desarrolladores, modifica considerablemente los requerimientos después de haberse completado el análisis inicial de requerimientos.

Diseño conceptual y diseño técnico

Para transformar los requerimientos en un sistema que funcione, los diseñadores deben satisfacer tanto a los clientes como a los constructores de sistemas. El diseño es un proceso iterativo que consta de dos partes. Primero se elabora un diseño conceptual o diseño de sistema que le dice al cliente, exactamente, qué hará dicho sistema. Una vez que el cliente aprueba el diseño conceptual, se vuelca este diseño en un documento mucho más detallado, el diseño técnico, que permite que los constructores comprendan el hardware y el software concretos que se necesitan para resolver el problema. El proceso es iterativo porque los diseñadores se mueven de una a otra de las diferentes actividades que abarcan la comprensión de requerimientos, la formulación de soluciones posibles, la comprobación de algunos aspectos de la solución, como la factibilidad, la presentación de alternativas a los clientes y la documentación del diseño para los programadores. El diseño conceptual se concentra en las funciones del sistema y el diseño técnico describe la forma que tomará el sistema.

El diseño conceptual describe el sistema en un lenguaje que el cliente puede comprender en lugar de usar términos técnicos o jerga de computación.

Un buen diseño conceptual debe tener las siguientes características:

- Se escribe en el lenguaje del cliente.
- No contiene expresiones en jerga técnica.
- Describe claramente las funciones del sistema.
- Es independiente de la implementación.
- Está vinculado con los documentos sobre requerimientos.

El diseño técnico describe la configuración del hardware, las necesidades de software, las interfaces de comunicación, la entrada y salida del sistema, la arquitectura de red y cualquier otro aspecto que incida en la transformación de los requerimientos en una solución para el problema del cliente. El diseño técnico incluye los siguientes ítems:

- Una descripción de los principales componentes de hardware t de sus funciones.
- La jerarquía y funciones de los componentes de software.
- Las estructuras de datos y el flujo de datos.

Descomposición y modularidad

Diseñar un sistema es determinar un conjunto de componentes y de interfaces entre componentes que satisfagan un conjunto específico de requerimientos. Todo método de diseño abarca un tipo de descomposición: a partir de una descripción de alto nivel de los elementos claves del sistema, y creando visiones a niveles inferiores para ver cómo las características y funciones del sistema se acomodarán en conjunto.

Wasserman sugiere que los diseños se crean en base a uno de los siguientes cinco enfoques:

- Descomposición modular: Se basa en la asignación de funciones a los componentes. El diseñador comienza con una descripción de alto nivel de las funciones que se van a implementar, y elabora explicaciones de menor nivel acerca de cómo esta organizado cada componente y cómo se relacionará con otros componentes.

- Descomposición orientada por datos: El diseño se basa en las estructuras externas de datos. La descripción de alto nivel presenta la estructura global de datos, y las descripciones de nivel más bajo proporcionan el detalle sobre los elementos de datos que estarán involucrados, y cómo estarán relacionados.
- Descomposición orientada por eventos: Se basa en los eventos que el sistema debe manejar y usa información sobre el modo en que estos eventos cambian el estado del sistema.
- Diseño "de afuera-hacia adentro": Es un enfoque de caja negra basado en las entradas en el sistema que realiza el usuario.
- Diseño orientado a objetos: Identifica clases de objetos y sus interrelaciones.

Independientemente del enfoque de diseño adoptado, cada tipo de descomposición separa el diseño en partes componibles denominadas módulos o componentes. Se dice que un sistema el modular cuando cada una de las actividades la realiza exactamente u único componente, y en donde están bien definidas cada una de sus entradas y salidas. Se dice que un componente está bien definido cuando todas las entradas en él son esenciales para su función, y todas las salidas son generadas por una de sus acciones.

Estilos arquitectónicos y estrategias

Shaw y Garlan sugieren que la arquitectura de software e el primer paso en la producción de un diseño de software. Con este enfoque pueden diferenciarse tres niveles de diseño:

- **Arquitectura** asocia las capacidades del sistema identificadas en la especificación de requisitos, con los componentes del sistema que las implementarán. Por lo general estos componentes son módulos y la arquitectura también describe las interconexiones entre ellos, la arquitectura define aperadores que crean sistemas a partir de subsistemas.
- **Diseño de código** comprende algoritmos y estructura de datos, y los componentes son primitivas del lenguaje de programación, tales como números, caracteres, punteros e hilos de control.
- **Diseño ejecutable** trata entre otros aspectos la asignación de memoria, los formatos de datos o la configuración de bits

Tuberías y filtros

Un componente de este tipo, tiene corriente de datos, denominadas tuberías para la entrada y la salida. La transformación de la entrada para producir la salida se inicia cuando el componente llamado filtro finaliza la lectura de la corriente de entrada.

Los sistemas de tipo tuberías y filtros tienen algunas propiedades importantes:

- Los diseñadores pueden entender todos los efectos del sistema sobre la entrada y salida, así como la composición de los filtros.
- Dado que pueden vincularse dos filtros cualesquiera, son fácilmente reutilizables entre otros sistemas.
- La evolución de los sistemas es simple, pueden incorporarse nuevos filtros y removerse los antiguos con relativa sencillez.
- Debido a la independencia de los filtros, los diseñadores pueden simular el comportamiento del sistema y analizar propiedades tales como el rendimiento.
- Se pueden ejecutar en forma concurrente.

Diseño orientado a objetos

Tiene dos características significativas: el objeto puede preservar la integridad de la representación de los datos, y la representación de datos puede ocultarse a los restantes objetos del sistema. Un objeto debe conocer la existencia de otros a fin de poder interactuar con ellos. Esta dependencia significa que el cambio de identidad de un objeto requiere que todos los objetos que invocan al objeto modificado deben, a su vez, ser modificados.

Invocación implícita

Es determinado por eventos y se basa en el concepto de difusión. En lugar de invocar directamente a un procedimiento, un componente anuncia que se han producido uno o más eventos.

Este estilo de diseño se presenta especialmente para la reutilización de componentes provenientes de otros sistemas. Dado que cualquier componente puede registrarse para un evento, un componente reutilizado puede agregarse al sistema y registrarse a sí mismo, independientemente de otros componentes. La mayor desventaja de este estilo de diseño es que no hay garantía de que el componente responderá al evento.

Estratificación (modelos de capas)

Las capas son jerárquicas; cada una presenta servicios a la capa inmediatamente superior y actúa como cliente de la que queda encerrada. En algunos sistemas, todas las capas tienen acceso a una parte o la totalidad de los servicios de las otras capas; en otros, el acceso de una capa queda limitado solo a las adyacentes.

El acceso de los usuarios a las diferentes capas del sistema dependerá de las pautas y necesidades expresadas en los requerimientos.

Las arquitecturas estratificadas se valen de la noción de abstracción. Es decir, se considera que cada capa tiene un nivel superior de abstracción, y los diseñadores pueden utilizar las capas para descomponer un problema en una secuencia de pasos más abstractos.

El inconveniente es que no siempre es fácil estructurar un sistema en capas, las múltiples capas de abstracción no se vuelven evidentes cuando se examina el conjunto de requerimientos.

Repositorios

Un repositorio tiene dos clases se componentes: un almacenamiento de datos y un conjunto de componentes que operan sobre el almacenamiento, recuperar o almacenar información. En una base de datos convencional. Las transacciones, en forma de corriente de entrada dispara la ejecución de los procesos. En un tablero de control o pizarra el almacén central controla el disparo de los procesos.

Muchos sistemas están organizados como repositorios: bibliotecas de componentes reutilizables, grandes bases de datos y motores de búsqueda. Una ventaja de este tipo de arquitectura es su accesibilidad. Esta característica conlleva una desventaja: los datos compartidos deben adoptar una forma aceptable para todas las fuentes de conocimiento, aun cuando estas fuentes son en sí mismas radicalmente diferentes.

Intérpretes

Un intérprete toma una cadena de caracteres denominada pseudocódigo y la convierte en código verdadero que luego se ejecuta. El intérprete esta compuesto de cuatro partes:

- Una memoria para contener el pseudocódigo a ser interpretado.
- Un motor de interpretación para convertir el pseudocódigo y simular el programa que representa.
- Es estado actual del motor de interpretación.
- El estado actual del programa que se está simulando.

Control de procesos

El propósito de un sistema de control de procesos es mantener las propiedades específicas de las salidas del proceso dentro o cerca de los valores de referencia especificados, denominados puntos fijos o valores de calibración.

El sistema de control basado en software más común comprende uno de los dos tipos de lazo cerrado: retroalimentación o prealimentación. Un sistema retroalimentado mide una variable controlada, y ajusta el proceso en concordancia para mantener el valor de la variable cerca o dentro del valor de calibración. En la prealimentación el sistema intenta anticipar los futuros

efectos sobre la variable controlada, midiendo otras variables de proceso que pueden actuar como buenos indicadores.
Shaw y Garlan recomiendan que la arquitectura separe las tres partes del lazo de control:

- Elementos computacionales, separando procesos de política de control.
- Elementos de datos, incluyendo las variables de proceso, puntos de calibración y sensores que van a usarse.
- Esquema del lazo de control.

Problemas en la creación del diseño

Modularidad y niveles de abstracción

En un diseño modular, los componentes tienen entradas y salidas claramente definidas y cada componente tiene un propósito claramente establecido. Los componentes modulares están organizados en una jerarquía, como resultado de la descomposición o abstracción.
Los niveles de abstracción ayudan a comprender el problema planteado por el sistema y la solución que propone el diseño.
Combinando componentes modulares y niveles de abstracción se obtienen varias vistas diferentes del sistema.
Otro beneficio de la modularidad es que permite diseñar diferentes componentes de forma distinta.

Diseño colaborativo

Uno de los mayores problemas para llevar adelante un diseño es equipo es encarar las diferencias en experiencia, comprensión y preferencias personales de los integrantes del equipo.
Yourdon identifica cuatro etapas en el desarrollo distribuido:

- En la primera etapa, el proyecto se realiza en un único sitio con desarrolladores provenientes de otros países, pero instalados en el lugar.
- En la segunda etapa, los analistas locales determinan los requerimientos del sistema.
- En la tercera etapa, desarrolladores externos construyen productos y componentes genéricos que son usados mundialmente.
- En la cuarta etapa, los desarrolladores externos construyen productos que saquen ventajas de sus áreas individuales de experiencia.

Diseño de la interfaz de usuario

Marcus puntualiza que la interfaz debe atender algunos elementos clave:

- Metáforas.
- Un modelo mental.
- Reglas de navegación para el modelo.
- Aspecto.
- Sensación.

A fin de diseñar interfaces efectivas y confortables, debemos considerar dos cuestiones claves: cultura y preferencia.
Cuestiones culturales. Los prototipos pueden utilizarse en la etapa de diseño para analizar preferencias y determinar qué tipos de interfaz son factibles y satisfacen los requerimientos de rendimiento y confiabilidad. Como el software es utilizado a nivel mundial, se deben considerar las creencias, valores, normas, tradiciones, costumbres y mitos de quienes utilizan los sistemas.
Para hacer que los sistemas sean miticulturales, las interfaces podrán diseñarse en dos etapas. Primero se eliminan todas las referencias culturales específicas o brechas, para hacer que la interfaz resulte tan internacional como sea posible.
La segunda etapa toma el diseño ya libre de brechas y lo amolda a las culturas que lo van a utilizar.
Es importante recordar que la cultura es determinada no sólo por la nacionalidad, sino también por la región, sexo, profesión, edad o corporación.

- ***Preferencias del usuario.*** Algunos aspectos del diseño dependen de las preferencias del usuario, ya sea solo o como miembro de un grupo de trabajo. Por ejemplo, Marcus (1993) propone tres interfaces, cada una adaptada a una audiencia diferente. La interfaz 1estaba pensada para varones intelectuales, europeos angloparlantes. La interfaz 2 se suponía apropiada para una audiencia de mujeres estadounidenses blancas, y la tercera interfaz era para consumidores angloparlantes con preferencias por diseños internacionales.

Teasley et al. (1994) probaron las tres interfaces de Marcus para determinar si satisfacían las preferencias de las audiencias que Marcus predecía. En la opinión de los encuestados, 40 de los 89 sujetos en estudio pensaron que la interfaz 2 era para europeos y 41 de los 89 opinaron que la interfaz 1 estaba pautada para mujeres.

Por ende, parece no haber una interfaz universal que pueda aplicarse a cualquier cultura, y puede ser difícil describir las pautas de diseño que asegurarán que los usuarios estarán satisfechos con las interfaces de nuestro sistema.

Los resultados de este y otros estudios enfatizan la importancia de hacer prototipos en conjunto con la audiencia destinataria para el sistema que se está diseñando.

Pautas para determinar las características de la interfaz de usuario.

La elección de las características de diseño en una interfaz de usuario entraña múltiples negociaciones. Se sugiere que se formulen las elecciones de diseño en términos de un espacio de diseño, es decir, cada negociación refleja al menos dos dimensiones de la elección.

Cuestiones a considerar en negociaciones de análisis

- ✓ Tratamiento de eventos externos
- ✓ Adaptabilidad al usuario
- ✓ Adaptabilidad transversal de Interfaz de Usuario(IU) y dispositivos
- ✓ Organización del sistema de computadora
- ✓ Tipo básico de interfaz
- ✓ Portabilidad de la aplicación a través de la IU
- ✓ Nivel de abstracción de la interfaz de aplicación
- ✓ Variabilidad del dispositivo abstracto
- ✓ Notación para definir la interfaz de usuario
- ✓ Fundamento de la comunicación
- ✓ Mecanismo de control de hilos de proceso

Concurrencia

En muchos sistemas las acciones se llevan a cabo concurrentemente, es decir, no secuencialmente. Los sistemas secuenciales por lo común utilizan una corriente simple de ejecución para controlar eventos, pero los sistemas concurrentes exigen diseños mucho más complejos.

Una forma de manejar la concurrencia es establecer una cantidad de tiempo asignado para la ejecución de cualquier acción. De esta forma, una temporización cuidadosa puede asegurar que las acciones no interfieran unas con otras. Sin embargo, la temporización no siempre está bajo el control del sistema, especialmente en sistemas de tiempo real que reaccionan a eventos externos.

Para tratar este tipo de problemas se utilizan técnicas para sincronización de procesos concurrentes. La sincronización es un método que permite que dos actividades ejecuten concurrentemente sin interferir una con otra.

Exclusión mutua, es una forma común de sincronizar procesos; garantiza que, cuando un proceso está accediendo a un elemento de datos, ningún otro proceso puede afectar a ese elemento. Si una operación prueba el valor del estado de un objeto, entonces este objeto debe quedar bloqueado de modo que el estado no cambie entre el instante que se hace la comprobación y el instante en que se ejecuta la acción basada en el valor producido por la prueba.

También puede usarse un ***criterio de prioridad de componentes o de procesos*** para resolver conflictos de concurrencia. Aquel con más alta prioridad puede ganar la batalla entre dos procesos o componentes, bloqueando efectivamente al otro hasta que se complete la acción de más alta prioridad.

Idealmente se debe aspirar a diseñar los sistemas de manera que sean correctos, sin depender de la temporización de las solicitudes. Afortunadamente, hay dos maneras de hacer esto: monitores y guardianes.

Monitores. Un monitor es un objeto abstracto o componente que controla la exclusión mutua de un proceso particular. Para prevenir problemas de temporización el monitor habitualmente se complementa con un probador de condición, para garantizar que las condiciones son las correctas para invocar el proceso solicitado.

Guardianes. Un guardián es una tarea que se ejecuta permanentemente; su único propósito es controlar el acceso a un recurso encapsulado. Como el monitor, un guardián también incluye un probador de condición que lo asiste en la toma de decisiones de control de acceso.

Patrones de diseño y reutilización

Con frecuencia se diseñan y construyen sistemas que son similares en algunos aspectos a los sistemas que se han construido anteriormente. O bien se pueden diseñar y construir una serie de aplicaciones que tengan funcionalidades similares pero destinadas a ejecutarse en ambientes diferentes. Por lo tanto se puede sacar ventaja de este comunitarismo entre sistemas y no se necesita desarrollar cada uno desde el borrador.

Una manera de identificar estos comunitarismos es observar la existencia de *patrones de diseño*. Luego, cuando se construya un sistema similar se podrán reutilizar los patrones, así como el código, pruebas y documentos relacionados con ellos.

Un **Patrón de Diseño** nombra, abstrae e identifica los aspectos claves de una estructura común de diseño que la hacen útil para la creación de un… diseño reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades.

Características de un buen diseño

Los diseños de calidad superior deben tener características que dan como resultado productos de calidad: facilidad de comprender, de implementar, de probar, de modificar y la correcta traducción de las especificaciones de requerimientos. La capacidad de modificación es especialmente importante, dado que los cambios que se hace necesarios para corregir defectos a veces producen un cambio del diseño.

Independencia de los componentes

La abstracción y el ocultamiento de información nos permiten examinar las formas en que cada componente se relaciona con algún otro en el diseño global. En la mayoría de los diseños de trata de que los componentes sean independientes unos de otros. No solamente es más fácil entender como trabaja un componente si no está ligado intrincadamente a otros, si no que un componente independiente es mucha más fácil de modificar.

Para reconocer y medir el grado de independencia de los componentes de un diseño se aplican dos conceptos: acoplamiento y cohesión.

Acoplamiento. Se dice que dos componentes están altamente acoplados cuando existe mucha dependencia entre ellos. Los componentes poco acoplados tienen algunas dependencias, para las interconexiones entre ellos son débiles. Los componentes no acoplados no tienen interconexiones con el resto.

La dependencia de un componente respecto de otro puede establecerse de distintas formas, por lo que el acoplamiento depende de varios aspectos:

- ✓ Las referencias hechas de un componente a otro

- ✓ La cantidad de datos pasados de un componente a otro
- ✓ El grado de control que un componente tiene sobre otro
- ✓ El grado de complejidad de la interfaz entre los componentes

En consecuencia, podemos medir el acoplamiento en función de un rango de dependencia que de la dependencia completa a la completa independencia.
Nuestra meta no necesariamente es la completa independencia sino mantener el menor grado de acoplamiento posible. Es decir, deseamos minimizar la dependencia entre componentes. Si un elemento es afectado por una acción del sistema, siempre es conveniente conocer cuál componente cusa el efecto en un momento dado. Tener esta información permite cambiar una porción del diseño del sistema, y alterar el diseño lo menos posible.
Algunos tipos de acoplamiento son menos convenientes que otros. El menos conveniente se produce cuando un componente de hecho modifica a otro. Entonces, el componente modificado es completamente dependiente del que lo modifica. Se denomina acoplamiento de contenido.
Cohesión. En contraste con la interdependencia entre componentes, la cohesión se refiere al grado de "adhesivo" interno con el que se ha construido el componente. Cuando más cohesivo es un componente, más relacionadas están las partes internas de él, tanto entre ellas como en relación al propósito global.
El peor grado de cohesión, la cohesión coincidental, ocurre cuando las partes de un componente no tiene relación alguna entre si. En este caso, las funciones, procesos o datos no racionados se encuentran en un mismo componente por razones de conveniencia o simplemente por que si.
El siguiente nivel de cohesión es la cohesión lógica, en donde algunas funciones o elementos de datos relacionados lógicamente están puestos en el mismo componente.
A veces un componente se utiliza para inicializar un sistema o un conjunto de variables. Este componente realiza varias funciones en sí sólo están relacionadas por el momento en que ocurren, o sea que su cohesión es temporal.
A menudo las funciones deben juntarse en un determinado orden. Cuando las funciones se agrupan en un mismo componente, justamente para asegurar el orden previsto, el componente tiene cohesión procedimental. Como alternativa, podemos asociar determinadas funciones debido a que operan sobre, o producen, el mismo conjunto de datos, tienen cohesión comunicativa.
Nuestro ideal es la Cohesión Funcional donde cada elemento de proceso es esencial para la realización de una única función y todos los elementos esenciales están contenidos en un único componente. Un componente funcionalmente cohesivo no sólo realiza la función para la cual ha sido diseñada, sino que realiza esa función y nada más.
Identificación y tratamiento de excepciones
Debemos diseñar defensivamente, tratando de anticiparnos a situaciones que podrían derivar en problemas en el sistema. No es fácil diseñar defensivamente. La especificación de requerimientos nos indica qué es lo que se supone que debe hacer el sistema, pero normalmente no explicita lo que el sistema no hace.
Excepciones, esto es, situaciones que se saben contrarias a lo que se espera realmente que haga el sistema. Entonces, en el diseño se incluirá el manejo de excepciones de modo que el sistema dirija cada excepción en una forma satisfactoria que no degrade las funciones del sistema.

- Las excepciones más comunes son:
- Fracaso al proporcionar un servicio
- Proporcionar un servicio o datos erróneos
- Corrupción de datos

Cada una de las excepciones que se identifican puede manejarse según uno de los siguientes criterios:

1. Reintentar, se restaura el sistema a su estado previo y se intenta realizar el servicio utilizando una estrategia diferente.
2. Corregir, se restaura el sistema a sus estado previo y se intenta realizar el servicio utilizando la misma estrategia.
3. Informar, se restaura el sistema a su estado previo, se informa el problema a un componente de tratamiento de error y no se proporciona el servicio.

Por lo tanto, para cada servicio que deseamos que el sistema realice, debemos identificar las formas en que puede fallar, así como la forma de rescatarlo de la falla.

El sistema final es la síntesis de los resultados generados por los prototipos individuales. Un prototipo desechable pensado con la sola finalidad de contribuir a determinar la factibilidad u otras características particulares en un sistema de mayor envergadura, tras lo cual simplemente se lo descarta. Siempre que los clientes entiendan que el prototipo es sólo un modelo exploratorio y no un producto refinado, el prototipado puede ser útil, tanto para los clientes como para los desarrolladores, ayudan a comprender lo que el sistema debe hacer. Con un gran cuidado en la definición y desarrollo de los componentes de más alto nivel, un prototipo rápido puede responder cuestiones acerca del diseño, al mismo tiempo que proporciona bloques constructivos para el sistema final. Incorpora especificación, diseño, implementación y prueba en un solo paso. El inconveniente es que este proceso debe ser rápido para tener algún valor. El desempeño de operación de los productos desarrollados usando prototipos es tan bueno como los desarrollados utilizando técnicas tradicionales de diseño.

Análisis de árbol de defectos

Un método originalmente desarrollado para el programa milístico US Minuteman, razona sobre el diseño, ayudándonos a descomponerlo y a ver las situaciones que podrían llevar a una falla. Se construyen para representar el camino lógico que lleva del efecto a su causa. Se usan para dar soporte a la corrección de defectos o a la tolerancia, dependiendo de la estrategia de diseño que se haya elegido. Se comienza el análisis identificando las posibles fallas. Se consideran las fallas que podrían ser afectadas por el diseño, la operación y aun el mantenimiento. Se utiliza un conjunto de palabras guía. Se pueden seleccionar palabras guía y listas de control propias, basadas en el dominio de aplicación donde debe trabajar el sistema. A continuación se construye un gráfico cuyos nodos son fallas, ya sea de componentes únicos, de funciones del sistema o del sistema completo. Los bordes del grafo indican las relaciones ente los nodos. Cada borde se rotula con un descriptor lógico: and, si ambos componentes pueden fallar, or si uno u otro pueden hacerlo. A veces un borde se rotula n_d_m, si el sistema involucra m componentes redundantes, donde le componente n que falle indica la designación de la falla. Cada nodo debe representar un evento independiente. Se puede hacer la búsqueda de distintos tipos de debilidades del diseño: puntos singulares de falla, donde la seguridad o integridad del sistema dependen de un único componente; incertidumbre, donde no hay suficientes restricciones sobre los valores de las variables o condiciones a las cuales ramificar; ambigüedad, falta de componentes. A partir de este árbol de defectos se puede elaborar otro grafo conocido como árbol de grupos de corte. Sirve para determinar los puntos singulares de falla, especialmente cuando los árboles de defectos son complejos y difíciles de analizar por simple observación. Se aplican las siguientes reglas: 1) trabajando de arriba hacia abajo, asignar el nodo tope del árbol de grupo de corte, a la primera compuerta lógica en el tope del árbol de defectos. 2) si la encontrada es una compuerta or, dividir el árbol de grupo de corte en dos; si es una compuerta and, se incluye un nodo de composición construido de nodos hijos. 3) continuar hasta que todos los nodos hoja sean eventos básicos o nodos composición de eventos básicos.

Un grupo de corte es el conjunto de nodos hoja del árbol de grupos de corte con las duplicaciones removibles. Los conceptos pueden aplicarse a cualquier sistema, hardware o software. La

descomposición de diseño por análisis de árbol de defecto para software pude ser más precisa. Toda estructura de secuencia, decisión e iteración pude ser convertida a su representación equivalente como árbol de grupo de corte, y se obtiene un gran grupo de corte como derivación de este árbol. Conociendo los puntos de falla en nuestro diseño, podemos rediseñarlo para reducir vulnerabilidades. Cuando se encuentra un defecto en el diseño pueden adoptarse distintas alternativas de solución: eliminarlo, agregar componentes o condiciones para prevenir las condiciones de entrada que hacen que el defecto se manifieste, agregar componentes que harán la recuperación del daño que la falla provocará.

Los árboles de defectos también son útiles para calcular la probabilidad de ocurrencia de una determinada falla. Sin embargo, la construcción de los grafos consume tiempo. Muchos sistemas involucran muchas dependencias, de modo que resulta muy difícil detectar inconsistencias, así como es difícil centrarse solamente en las partes mas criticas del diseño, a menos que el acoplamiento sea realmente muy reducido. La cantidad y clases de precondiciones que son necesarias para determinar cada falla son intimidantes y no siempre fáciles de localizar, y no existen mediciones que ayuden a clasificarlas.

Evaluación y validación del diseño.

Una vez diseñado un sistema, se lo comprueba en dos formas diferentes. En primer lugar se asegura que el diseño satisface todos los requerimientos especificados por el cliente. Se conoce como validación del diseño. La verificación involucra asegurar que se han incorporado las características del buen diseño. Las herramientas automatizadas pueden contribuir tanto a la validación como a la verificación.

Validación matemática.

Algunos investigadores han logrado imponer cierto rigor matemático en este tipo de validación. Asociado a cada proceso existe un conjunto de entradas, un conjunto de salidas esperadas y un conjunto de aserciones acerca del proceso mismo. Para cada uno de estos procesos se demuestra que: si el conjunto de entradas está formulado correctamente, se transforma propiamente en el conjunto de salidas esperadas, el proceso termina sin fallas. Ese procedimiento “prueba” que el diseño es correcto. Sin embargo al usarlo puede resultar costoso y consumir mucho tiempo. Su aplicación se limita a las partes más críticas.

Medición de la calidad del diseño.

Briand, Devanbu y Melo trabajan en ideas para proponer formas de medir el acoplamiento. Hacen notar que el acoplamiento en C++ puede bastarse en tres características diferentes: relación, sitio y tipo. Para cada una de las clases de un diseño, definen métricas que hacen el conteo de interacciones entre la clase y las horas clases o métodos. Luego, analizan la relación ente el tipo de acoplamiento y la naturaleza de los defectos encontrados, utilizando información empírica sobre los defectos y fallas resultantes del sistema. Card y Glass puntualizan que la complejidad del diseño realmente comprende dos aspectos: la complejidad intrínseca del componente y la complejidad de las relaciones entre componentes. La medida final de la complejidad es la suma: $C=S+D$ donde $S=(1/n)\sum f^2(i)$, $D=V(i)/[f(i)+1]$; S es la complejidad estructural, D la complejidad de datos, f(i) abanico de salida del componente i, V(i) numero de variables de entrada y salida del componente i, n es el numero de componentes.

Revisiones del diseño.

Se realiza en 3 etapas en correspondencia con los pasos del proceso de diseño. En primer término se hará una revisión del diseño preliminar, para revisar el diseño conceptual conjuntamente con los clientes y usuarios. A continuación, en una revisión crítica de diseño se presentará el diseño técnico a otros desarrolladores para verificar sus detalles, antes de proceder a la implementación. Finalmente se hará una revisión del diseño de programas para que los programadores tengan

alguna realimentación acerca del diseño antes de la implementación. La meta es asegurar que se está construyendo lo que los clientes desean y necesitan.

Documentando el diseño.

Deben contener una sección denominada justificación racional del diseño, donde se delinean las cuestiones críticas y compromisos que fueron considerados en la generación del diseño. Una de la secciones debería indicar cómo interactúan los usuarios con el sistema, incluyendo: menúes y otros formatos de presentaciones en pantalla, interfaces hombre-máquina: teclas de función, descripciones de pantallas sensibles al tacto, esquemas de teclados y el uso del mouse, etc.