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

DISEÑO DE SISTEMAS

TAREA #4

PRESENTADO POR:

DAVID MARTINEZ PEÑA

ICO 9

ATLACOMULCO, MEXICO, A 27 DE OCTUBRE DE 2008.

CONCEPTOS Y PRINCIPIOS DE DISEÑO

El objetivo de los diseñadores es producir un modelo o representación de una entidad que será construida a posteriori. En cualquier proceso de diseño existen dos fases importantes: la diversificación y la convergencia. La diversificación es la ***adquisición*** de un repertorio de alternativas, de un material primitivo de diseño. Durante la convergencia, el diseñador elige y combina los elementos adecuados y extraídos de este repertorio para satisfacer los objetivos del diseño, de la misma manera a como se establece en el documento de los requisitos, y de la manera en que se acordó con el cliente. La segunda fase es la ***eliminación*** gradual de cualquier configuración de componentes excepto de una en particular, y de aquí la creación del producto final.

La diversificación y la convergencia combinan intuición y juicio en función de la experiencia en construir entidades similares; un conjunto de principios y/o heurística que proporcionan la forma de guiar la evolución del modelo; un conjunto de criterios que posibilitan la calidad que se va **a** juzgar, y un proceso de iteración que por Último conduce a una representación final de diseño.

DISEÑO DEL SOFTWARE E INGENIERIA DEL SOFTWARE

El diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas -diseño, generación de código y pruebas- que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que dé lugar por último a un software de computadora validado.

Los requisitos del software, manifestados por los modelos de datos funcionales y de comportamiento, alimentan la tarea del diseño. Mediante uno de los muchos métodos de diseño (que se abarcarán en capítulos posteriores) la tarea de diseño produce un diseño de datos, un diseño arquitectónico, un diseño de interfaz y un diseño de componentes.

El *diseño de datos* transforma el modelo del dominio de información que se crea durante el análisis en las estructuras de datos que se necesitarán para implementar el software. Los objetos de datos y las relaciones definidas en el diagrama relación entidad y el contenido de datos detallado que se representa en el diccionario de datos proporcionan la base de la actividad del diseño de datos.

El *diseño arquitectónico* define la relación entre los elementos estructurales principales del software, los patrones de diseño que se pueden utilizar para lograr los requisitos que se han definido para el sistema, y las restricciones que afectan a la manera en que se pueden aplicar los patrones de diseño arquitectónicos**.** La representación del diseño arquitectónico -el marco de trabajo de un sistema basado en computadora- puede derivarse de la especificación del sistema, del modelo de análisis y de la interacción del subsistema definido dentro del modelo de análisis.

El *diseño de la interfaz* describe la manera de comunicarse el software dentro de sí mismo, con **sis**temas que interoperan dentro de él y con las personas que lo utilizan. Una interfaz implica un flujo de información (por ejemplo, datos y/o control) y un tipo específico de comportamiento. Por tanto, los diagramas de flujo de control y de datos proporcionan gran parte de la información que se requiere para el diseño de la interfaz.

El *diseño a nivel de componentes* transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software. La información que se obtiene de EP, EC **y** de DTE sirve como base para el diseño de los componentes.

La importancia del diseño del software se puede describir con una sola palabra –*calidad.* El diseño es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado. El diseño del software sirve como fundamento para todos los pasos siguientes del soporte del software y de la ingeniería del software. Sin un diseño, corremos el riesgo de construir un sistema inestable -un sistema **que** fallará cuando se lleven a cabo cambios; un sistema que puede resultar difícil de comprobar; **y** un sistema cuya calidad

no puede evaluarse hasta muy avanzado el proceso, sin tiempo suficiente y con mucho dinero gastado en él-.

EL PROCESO DE DISEÑO

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un «plano» para construir el software. El diseño se representa a un nivel alto de abstracción -un nivel que puede rastrearse directamente hasta conseguir el objetivo del sistema específico y según unos requisitos más detallados de comportamiento, funcionales y de datos-. A medida que ocurren las iteraciones del diseño, el refinamiento subsiguiente conduce a representaciones de diseño a niveles de abstracción mucho más bajos.

Diseño y calidad del software

La calidad de la evolución del diseño se evalúa con una serie de revisiones técnicas formales o con las revisiones de diseño sugiere tres características que sirven como guía para la evaluación de un buen diseño: el diseño deberá implementar todos los requisitos explícitos del modelo de análisis, y deberán ajustarse a todos los requisitos implícitos que desea el cliente; el diseño deberá ser una guía legible y comprensible para aquellos que generan código y para aquellos que comprueban y consecuentemente, dan soporte al software; el diseño deberá proporcionar una imagen completa del software, enfrentándose a los dominios de comportamiento, funcionales y de datos desde una perspectiva de implementación.

1. Un diseño deberá presentar una estructura que se haya creado mediante Patrones de diseño reconocibles, que esté formada por componentes que exhiban características de buen diseño, que se puedan implementar de manera evolutiva, facilitando así la implementación Y la comprobación. **2.** Un diseño deberá ser modular. 3. Un diseño deberá contener distintas representaciones de datos, arquitectura, interfaces y componentes (módulos). 4. Un diseño deberá conducir a estructuras de datos adecuadas para los objetos que se van a implementar y que procedan de patrones de datos reconocibles. *5.* un diseño deberá conducir a componentes que. 6. Un diseño deberá conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y con el entorno externo. 7. un diseño deberá derivarse mediante un método repetitivo y controlado por la información obtenida durante el análisis de los requisitos del software.

La evolución del diseño del software

Es un proceso continuo que ha abarcado las últimas cuatro décadas. El primer trabajo de diseño se concentraba en criterios para el desarrollo de programas modulares y métodos para refinar las estructuras del software de manera descendente. Los aspectos procedimentales de la definición de diseño evolucionaron en una filosofía denominada *estructurada.* Un trabajo posterior propuso para la conversión del flujo de datos en una definición de diseño. Enfoques de diseño más recientes hacia la derivación de diseño proponen un método orientado a objetos. Hoy en día, se ha hecho hincapié en un diseño de software basado en la arquitectura del software.

Independientemente del modelo de diseño que se utilice, un ingeniero del software deberá aplicar un conjunto de principios fundamentales y conceptos básicos para el diseño a nivel de componentes, de interfaz, arquitectónico y de datos. Estos principios y conceptos se estudian en la sección siguiente.

PRINCIPIO DEL DISEÑO

El *proceso* de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario. El *modelo* de diseño es el equivalente a los planes de un arquitecto para una casa. Comienza representando la totalidad de todo lo que se va a construir (por ejemplo, una representación en tres dimensiones de la casa) y refina lentamente lo que va a proporcionar la guía para construir cada detalle (por ejemplo, el diseño de fontanería).

En el proceso de diseño no deberá utilizarse «orejeras ». Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requisitos del problema, los recursos disponibles para realizar el trabajo y los conceptos de diseño.
El diseño deberá poderse rastrear hasta el modelo de análisis. Dado que un solo elemento del modelo de diseño suele hacer un seguimiento de los múltiples requisitos, es necesario tener un medio de rastrear cómo se han satisfecho los requisitos por el modelo de diseño.
El diseño no deberá inventar nada que ya esté inventado. Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales probablemente ya se han encontrado antes.
Estos patrones deberán elegirse siempre como una alternativa para reinventar. Hay poco tiempo y los recursos son limitados. El tiempo de diseño se deberá invertir en la representación verdadera de ideas nuevas y en la integración de esos patrones que ya existen.
El diseño deberá «minimizar la distancia intelectual » entre el software y el problema como si de la misma vida real se tratara. Es decir, la estructura del diseño del software (siempre que sea posible) imita la estructura del dominio del problema.
El diseño deberá presentar uniformidad e integración. Un diseño es uniforme si parece que fue una persona la que lo desarrolló por completo. Las reglas de estilo y de formato deberán definirse para un equipo de diseño antes de comenzar el trabajo sobre el diseño. Un diseño se integra si se tiene cuidado a la hora de definir interfaces entre los componentes del diseño.
El diseño deberá estructurarse para admitir cambios. Los conceptos de diseño estudiados en la sección siguiente hacen posible un diseño que logra este principio.
El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes. Un software bien diseñado no deberá nunca explotar como una <<bomba». Deberá diseñarse para adaptarse a circunstancias inusuales, y si debe terminar de funcionar, que lo haga de forma suave.
El diseño no es escribir código y escribir código no es diseñar. Incluso cuando se crean diseños procedimentales para componentes de programas, el nivel de abstracción del modelo de diseño es mayor que el código fuente.
El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo.
El diseño deberá revisarse para minimizar los errores conceptuales (semánticos). A veces existe la tendencia de centrarse en minucias cuando se revisa el diseño, olvidándose del bosque por culpa de los árboles. Un equipo de diseñadores deberá asegurarse de haber afrontado los elementos conceptuales principales antes de preocuparse por la sintaxis del modelo del diseño.
Losfactores de calidad externos son esas propiedades del software que pueden ser observadas fácilmente por los usuarios. Los *factores de calidad internos* tienen importancia para los ingenieros del software. Desde una perspectiva técnica conducen a un diseño de calidad alta. Para lograr los factores de calidad internos, el diseñador deberá comprender los conceptos de diseño básicos.

CONCEPTOS DEL DISEÑO

Durante las últimas cuatro décadas se ha experimentado la evolución de un conjunto de conceptos fundamentales de diseño de software. Aunque el grado de interés en cada concepto ha variado con los años, todos han experimentado el paso del tiempo. Cada uno de ellos proporcionará la base de donde el diseñador podrá aplicar los métodos de diseño más sofisticados.

Abstracción

Cuando se tiene en consideración una solución modular a cualquier problema, se pueden exponer muchos *niveles de abstracción.* En el nivel más alto de abstracción, la solución se pone como una medida extensa empleando el lenguaje del entorno del problema. En niveles inferiores de abstracción, se toma una orientación más procedimental. La terminología orientada a problemas va emparejada con la terminología orientada a la implementación en

un esfuerzo por solucionar el problema. Finalmente, en el nivel más bajo de abstracción, se establece la solución para poder implementarse directamente.

A medida que vamos entrando en diferentes niveles de abstracción, trabajamos para crear abstracciones procedimentales y de datos. Una *abstracción procedimental* es una secuencia nombrada de instrucciones que tiene una función específica y limitada. Un ejemplo de abstracción procedimental sería la palabra «abrir» para una puerta. «Abrir» implica una secuencia larga de pasos procedimentales (por ejemplo, llegar a la puerta; alcanzar **y** agarrar el pomo de la puerta; girar el pomo y tirar de la puerta; separarse al mover la puerta, etc.).

Una *abstracción de datos* es una colección nombrada de datos que describe un objeto de datos. En el contexto de la abstracción procedimental *abrir,* podemos definir una abstracción de datos llamada puerta**.** Al igual que cualquier objeto de datos, la abstracción de datos para **puerta** acompañaría a un conjunto de atributos que describen esta puerta (por ejemplo, tipo de puerta, dirección de apertura, mecanismo de apertura, peso, dimensiones). Se puede seguir diciendo que la abstracción procedimental *abrir* hace uso de la información contenida en los atributos de la abstracción de datos **puerta.**

La *abstracción de control* es la tercera forma de abstracción que se utiliza en el diseño del software. Al igual que las abstracciones procedimentales y de datos, este tipo de abstracción implica un mecanismo de control de programa sin especificar los datos internos. Un ejemplo de abstracción de control es el semáforo de sincronización [KAI83] que se utiliza para coordinar las actividades en un sistema operativo. El concepto de abstracción de control se estudia brevemente en el Capítulo 14.

Refinamiento

El *refinamiento paso a paso* es una estrategia de diseño descendente propuesta originalmente por Niklaus Wirth. El desarrollo de un programa se realiza refinando sucesivamente los niveles de detalle procedimentales.

Una jerarquía se desarrolla descomponiendo una sentencia macroscópica de función (una abstracción procedimental) paso a paso hasta alcanzar las sentencias del lenguaje de programación. En cada paso (del refinamiento), se descompone una o varias instrucciones del programa dado en instrucciones más detalladas.

Esta descomposición sucesiva o refinamiento de especificaciones termina cuando todas las instrucciones se expresan en función de cualquier computadora subyacente o de cualquier lenguaje de programación. De **la** misma manera que se refinan las tareas, los datos también se tienen que refinar, descomponer o estructurar, y es natural refinar el programa y las especificaciones de los datos en paralelo.

Modularidad

El concepto de modularidad se ha ido exponiendo desde hace casi cinco décadas en el software de computadora. La arquitectura de computadora expresa la modularidad; es decir, el software se divide en componentes nombrados y abordados por separado, llamados frecuentemente *módulos,* que se integran para satisfacer los requisitos del problema.

Se ha afirmado que <<la modularidad es el único atributo del software que permite gestionar un programa intelectualmente». El software monolítico (es decir, un programa grande formado por un único módulo) no puede ser entendido fácilmente por el lector. La cantidad de rutas de control, la amplitud de referencias, la cantidad de variables y la complejidad global hará que el entendimiento esté muy cerca de ser imposible. Para ilustrar este punto, tomemos en consideración el siguiente argumento basado en observaciones humanas sobre la resolución de problemas.

Arquitectura del software

La *arquitectura del software* alude a la «estructura global del software y a las formas en que la estructura proporciona la integridad conceptual de un sistema». En su forma más simple, la arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los

componentes. Sin embargo, en un sentido más amplio, los «componentes» se pueden generalizar para representar los elementos principales del sistema y sus interacciones.
Propiedades estructurales. Este aspecto de la representación del diseño arquitectónico define los componentes de un sistema (por ejemplo, módulos, objetos, filtros) y la manera en que esos componentes se empaquetan e interactúan unos con otros.
Propiedades extra-funcionales. La descripción del diseño arquitectónico deberá ocuparse de cómo la arquitectura de dise*ño c*onsigue **los** requisitos para el rendimiento, capacidad, fiabilidad, seguridad, capacidad de adaptación y otras características del sistema.
Familias de sistemas relacionados. El diseño arquitectónico deberá dibujarse sobre patrones repetibles que se basen comúnmente en el diseño de familias de sistemas similares.
Los *modelos dinámicos* tratan los aspectos de comportamiento de la arquitectura del programa, indicando cómo puede cambiar la estructura o la configuración del sistema en función de los acontecimientos externos. **Los** *modelos de proceso* se centran en el diseño del proceso técnico de negocios que tiene que adaptar el sistema. Finalmente los *modelos funcionales* se pueden utilizar para representar la jerarquía funcional de un sistema.
Se ha desarrollado un conjunto de *lenguajes de descripción arquitectónica* (LDAs) para representar los modelos. Aunque se han propuesto muchos LDAs diferentes, la mayoría proporcionan mecanismos para describir los componentes del sistema y la manera en que se conectan unos con otros.

13.4.5. Jerarquía de control

La *jerarquia de control,* denominada también estructura de programa, representa la organización de los componentes de programa (módulos) e implica una jerarquía de control. No representa los aspectos procedimentales del software, ni se puede aplicar necesariamente a todos los estilos arquitectónicos.
Para representar la jerarquía control de aquellos estilos arquitectónicos que se avienen a la representación se utiliza un conjunto de notaciones diferentes. El diagrama más común es el de forma de árbol que representa el control jerárquico para las arquitecturas de llamada y de retorno. Con objeto de facilitar estudios posteriores de estructura, definiremos una serie de medidas y térmi**nos** simples. La *profundidad* **y** la *anchura* proporcionan una indicación de la cantidad de niveles de control y el *ámbito de control* global, respectivamente.
El *grado de salida* es una medida del número de módulos que se controlan directamente con otro módulo. El *grado de entrada* indica la cantidad de módulos que controlan directamente un módulo dado.
La jerarquía de control también representa dos características sutiles diferentes de la arquitectura del software: visibilidad y conectividad. La *visibilidad* indica el conjunto de componentes de programa que un componente dado puede invocar o utilizar como datos, incluso cuando se lleva a cabo indirectamente. La *conectividad* indica el conjunto de componentes que un componente dado invoca o utiliza directamente como datos.

División estructural

Si el estilo arquitectónico de un sistema es jerárquico, la estructura del programa se puede dividir tanto horizontal como verticalmente. La partición horizontal define ramas separadas de la jerarquía modular para cada función principal del programa. Lo *módulos de control,* representados con un sombreado más oscuro se utilizan para coordinar la comunicación entre ellos y la ejecución de las funciones. El enfoque más simple de la división horizontal define tres particiones -entrada, transformación de datos (frecuentemente llamado procesamiento) y salida-. La división horizontal de la arquitectura proporciona diferentes ventajas: proporciona software más fácil de probar conduce a un software más fácil de mantener propaga menos efectos secundarios proporciona software más fácil de ampliar Como las funciones principales se desacoplan las unas de las otras, el cambio tiende a ser menos complejo y las extensiones del sistema tienden a ser más fáciles de llevar a cabo sin efectos secundarios.

Estructura de datos

La *estructura de datos* es una representación de la relación lógica entre elementos individuales de datos. Como la estructura de la información afectará invariablemente al diseño procedimental final, la estructura de datos es tan importante como la estructura de programa para la representación de la arquitectura del software.

Procedimiento de software

La *estructura de programa* define la jerarquía de control sin tener en consideración la secuencia de proceso **y** de decisiones. El procedimiento de software se centra en el procesamiento de cada módulo individualmente.

El procedimiento debe proporcionar una especificación precisa de procesamiento, incluyendo la secuencia de sucesos, los puntos de decisión exactos, las operaciones repetitivas e incluso la estructura/organización de datos.

Ocultación de información

Ocultación significa que se puede conseguir una modularidad efectiva definiendo un conjunto de módulos independientes que se comunican entre sí intercambiando sólo la información necesaria para lograr la función del software. La abstracción ayuda a definir las entidades (o información).

DISEÑO MODULAR EFECTIVO

Independencia funcional

El concepto de *independencia funcional* es la suma de la modularidad y de los conceptos de abstracción y ocultación de información. La independencia se mide mediante dos criterios cualitativos: la cohesión y el acoplamiento. La *cohesión* es una medida de la fuerza relativa funcional de un módulo. El *acoplamiento* es una medida de la independencia relativa entre los módulos.

Cohesión

Un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento de software, lo cual requiere poca interacción con los procedimientos que se llevan a cabo en otras partes de un programa.

Acoplamiento

El acoplamiento es una medida de interconexión entre módulos dentro de una estructura de software. El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo, y los datos que pasan a tra**vés** de la interfaz.

HEURÍSTICA DE DISEÑO PARA UNA MODULARIDAD EFECTIVA

La estructura de programa se puede manipular de acuerdo con el siguiente conjunto de heurísticas:

1. *Evaluar la «primera iteración» de la estructura de programa para reducir al acoplamiento y mejorar la cohesión.* Una vez que se ha desarrollado la estructura del programa, se pueden explosionar o implosionar los módulos con vistas a mejorar la independencia del módulo. Un *módulo explosionado* se convierte en dos módulos o más en la estructura final de programa. *Un módulo implosionudo* es el resultado de combinar el proceso implicado en dos o más módulos.

II. *Intentar minimizar las estructuras con un alto grado de salida; esforzarse por la entrada a medida que aumenta la profundidad.* Todos los módulos están «planos», al mismo nivel y por debajo de un solo módulo de control. En general, una distribución más razonable de control se muestra en la estructura de la derecha. La estructura toma una forma oval, indicando la cantidad de capas de control y módulos de alta utilidad a niveles inferiores.

III. *Mantener el ámbito del efecto de un módulo dentro del ámbito de control de ese módulo.* El *ámbito del efecto* de un módulo *e* se define como todos lo otros módulos que se ven afectados por la decisión tomada en el módulo ***e.*** El ámbito de control del módulo *e* se compone de todos los módulos subordinados y superiores al módulo *e.*

IV. *Evaluar las interfaces de los módulos para reducir la complejidad y la redundancia,* y *mejorar la consistencia.* La complejidad de la interfaz de un módulo es la primera causa de los errores del software. Las interfaces deberán diseñarse para pasar información de manera sencilla y deberán ser consecuentes con la función de un módulo. La inconsistencia de interfaces (es decir, datos aparentemente sin relacionar pasados a través de una lista de argumentos u otra técnica) es una indicación de poca cohesión.

V. *Definir módulos cuya función se pueda predecir, pero evitar módulos que sean demasiado restrictivos.* Un módulo es predecible cuando se puede tratar como una caja negra; es decir, los mismos datos externos se producirán independientemente de los datos internos de procesamiento.

VI. *Intentar conseguir módulos de «entrada controlada)), evitando «conexiones patológicas».* Esta heurística de diseño advierte contra el acoplamiento de contenido.

EL MODELO DEL DISEÑO

Los principios y conceptos de diseño abordados en este capítulo establecen las bases para la creación del modelo de diseño que comprende representaciones de datos, arquitectura, interfaces y componentes. Al igual que en el modelo de análisis anterior al modelo, cada una de estas representaciones de diseño se encuentran unidas unas a otras **y** podrán sufrir un seguimiento hasta los requisitos del software.

DOCUMENTACION DEL DISEÑO

La *Especificación del diseño* aborda diferentes aspectos del modelo de diseño y se completa a medida que el diseñador refina su propia representación del software. En primer lugar, ***se*** describe el ámbito global del esfuerzo realizado en el diseño. La mayor parte de la información que se presenta aquí se deriva de la *Especificación del sistema* y del modelo de análisis *(Especificación de los requisitos del software).* A continuación, se especifica el diseño de datos. Se definen también las estructuras de las bases de datos, cualquier estructura externa de archivos, estructuras internas de datos y una referencia cruzada que conecta objetos de datos con archivos específicos. El diseño arquitectónico indica cómo se ha derivado la arquitectura del programa del modelo de análisis.

La *Especificación del diseño* contiene una referencia cruzada de requisitos. El propósito de esta referencia cruzada (normalmente representada como una matriz simple) es: (1) establecer que todos los requisitos se satisfagan mediante el diseño del software, y (2) indicar cuales son los componentes críticos para la implementación de requisitos específicos.

El primer paso en el desarrollo de la documentación de pruebas también se encuentra dentro del documento del diseño. Una vez que se han establecido las interfaces y la estructura de programa podremos desarrollar las líneas generales para comprobar los módulos individuales y la integración de todo el paquete. En algunos casos, esta sección se podrá borrar de la *Especificación del diseño.* La última sección de la *Especificación del diseño* contiene datos complementarios.

CONCLUSIONES

Al diseñar debemos de tomar en cuenta todos los procedimientos utilizados, así como la arquitectura de nuestro sistema a fin de obtener un software para evaluar la calidad. Se proponen muchos conceptos y principios para el diseño de software. Y esto sirve de guía a medida que seguimos con el proceso de diseño. Un concepto como la modularidad permite que el diseñador simplifique y reutilice los componentes del software.

No por querer terminar pronto el software haremos el diseño tan rápido como sea posible es necesario tomarnos un tiempo para revisarlo con cuidado, de este modo obtendremos un producto de buena calidad.