Fundamentos del diseño de software

El diseño es el primer paso de la fase de desarrollo de cualquier producto o sistema de ingeniería.

Definición de diseño según Taylor

"*Proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física*"

El diseño de software, al igual que los métodos de diseño de todas las ingenierías, cambian continuamente al aparecer nuevos métodos, mejores análisis y ampliar los conocimientos. El problema es que el diseño de software se encuentra en una etapa relativamente temprana en su evolución. La idea de realizar diseño de software en lugar de "programar", surgió a principios de los años 60, por lo que a las metodologías de diseño les falta la profundidad y la flexibilidad que tiene el diseño en otras ingenierías. Pero, ya existen técnicas de diseño de software para poder evaluar la calidad del software.

El objetivo de este tema es presentar los conceptos fundamentales que se pueden aplicar a todos los diseños de programas.

1. Ingeniería del software y diseño del software

Una vez que se han establecido los requisitos del software, el diseño es la primera de tres actividades técnicas: **diseño**, **codificación** y **prueba**. Cada actividad transforma la información de forma que al final se obtiene un software validado.

El diseño es técnicamente la parte central de la ingeniería del software. Durante el diseño se desarrollan, revisan y se documentan los refinamientos progresivos de las estructuras de datos, de la estructura del programa y de los detalles procedimentales. El diseño da como resultado representaciones cuya calidad puede ser evaluada.

Mediante algunas metodologías de diseño se realiza el diseño de datos, el diseño arquitectónico y el diseño procedimental.

- El **diseño de datos** transforma el modelo de campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software.
- El **diseño arquitectónico** define las relaciones entre los principales elementos estructurales del programa.
- El **diseño procedimental** transforma los elementos estructurales en una descripción procedimental del software.

A continuación, se genera el código fuente y, para integrar y validar el software, se llevan a cabo las pruebas.

Las fases de diseño, codificación y prueba absorben el 75% o más del coste de la ingeniería del software (excluyendo el mantenimiento). Es aquí donde se toman las decisiones que afectarán finalmente al éxito de la implementación del programa, y también, a la facilidad de mantenimiento que tendrá el software. Por tanto el diseño es un paso fundamental de la fase de desarrollo.

El diseño es la única forma mediante la que podemos traducir con precisión los requisitos del cliente en un producto o sistema acabado. El diseño de software es la base de todas las partes posteriores del desarrollo y de la fase de prueba, como muestra la figura 1.

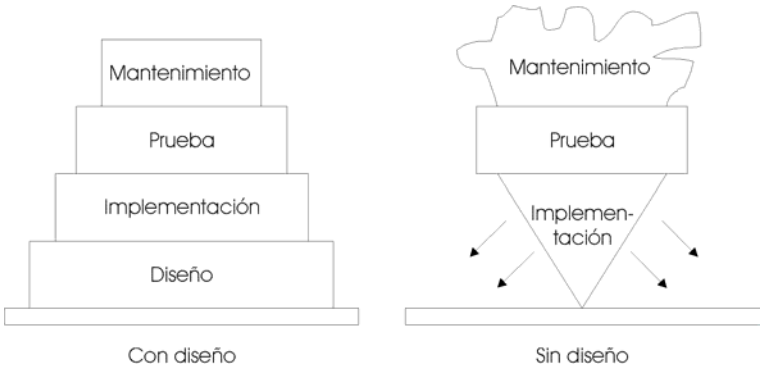

Figura 1. Importancia del diseño

Sin diseño, nos arriesgamos a construir un sistema inestable, un sistema que falle cuando se realicen pequeños cambios, un sistema que sea difícil de probar, un sistema cuya calidad no pueda ser evaluada hasta más adelante, cuando quede poco tiempo y ya sea haya gastado mucho dinero.

2. El proceso de diseño

El diseño del software es un proceso mediante el que se traducen los requisitos en una representación del software, que se acerca mucho al código fuente.

Desde el punto de vista de la gestión del proyecto, el diseño del software se realiza en dos etapas: el diseño preliminar y el diseño detallado.

- El **diseño preliminar** se centra en la transformación de los requisitos en los datos y la arquitectura del software.
- El **diseño detallado** se ocupa del refinamiento y de la representación arquitectónica que lleva a una estructura de datos refinada y a las representaciones algorítimicas del software.

Además del diseño de datos, del diseño arquitectónico y del desarrollo procedimental, muchas aplicaciones modernas requieren un **diseño de la interfaz**.

		Punto de vista gestión	
		Diseño preliminar	Diseño detallado
Punto de vista técnico	Diseño de datos	*	*
	Diseño arquitectónico	*	
	Diseño procedimental		*
	Diseño de la interfaz	*	*

Figura 2. Relación entre los puntos de vista de gestión y técnicos

2.1. DISEÑO Y CALIDAD DEL SOFTWARE

A lo largo del proceso de diseño, la calidad del diseño se evalúa mediante una serie de **revisiones técnicas formales (RTF)** que son una actividad de garantía del software cuyos objetivos son:

1) Descubrir los errores en la función, la lógica o la implementación de cualquier representación del software.
2) Verificar que el software alcanza sus requisitos.
3) Garantizar que el software se ha representado según los estándares establecidos.
4) Conseguir un software desarrollado de forma uniforme.
5) Hacer que los proyectos sean manejables.

Cada RTF se lleva a cabo mediante una reunión y sólo tendrá éxito si está bien planificada, controlada y atendida.

A continuación, se listan una serie de criterios para determinar la calidad del software.

1) Un diseño debe tener una organización jerárquica.
2) Un diseño debe ser modular, es decir, el software debe estar dividido en elementos que realicen funciones específicas.
3) Un diseño debe tener representaciones distintas y separadas de los datos y de los procedimientos.
4) Un diseño debe llevar a módulos que exhiban características funcionales independientes.
5) Un diseño debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el exterior.
6) Un diseño debe obtenerse mediante un método que sea reproducible y que esté dirigido por la información obtenida durante el análisis de requerimientos.

Un buen diseño de software no se consigue fácilmente, resultando de la aplicación de principios fundamentales de diseño, de una metodología sistemática y de una revisión exhaustiva.

2.2. CARACTERÍSTICAS COMUNES DE LAS METODOLOGÍAS DE DISEÑO

Independientemente de la metodología de diseño que se utilice, todas tienen varias características comunes:

1) Mecanismo para la traducción de requisitos en una representación de diseño.
2) Notación para representar los componentes funcionales y sus interfaces.
3) Heurísticas para el refinamiento y la partición.
4) Criterios para la valoración de la calidad.

Independientemente de la metodología de diseño que se utilice, el desarrollador tiene que aplicar una serie de conceptos fundamentales al diseño de datos, arquitectónico y procedimental.

3. Fundamentos del diseño

Los fundamentos del diseño ayudan al desarrollador de software a responder a estas preguntas:

- ¿Qué criterios puedo utilizar para dividir el software en componentes individuales?
- ¿Cómo se separan los detalles de una función o de la estructura de los datos de la representación conceptual del software?
- ¿Existen criterios uniformes que definan la calidad técnica de un diseño de software?

Cita de Michael A. Jackson

"*El principio de la sabiduría de un programador está en reconocer la diferencia entre obtener un programa que funcione y uno que funcione* ***correctamente***".

3.1. ABSTRACCIÓN

Cuando se considera una solución modular para cualquier problema, pueden formularse varios **niveles de abstracción**.

En el nivel superior de abstracción se establece una solución en términos generales, en lenguaje natural. En los niveles inferiores de abstracción se utiliza una orientación más procedimental. Por último, en el nivel más bajo de abstracción, se establece una solución, de forma que pueda implementarse directamente.

Cada paso de los procesos de la ingeniería del software es un refinamiento del nivel de abstracción de la solución software. Conforme nos movemos desde los preliminares hacia el diseño detallado, se reduce el nivel de abstracción. Finalmente, el nivel más bajo de abstracción se alcanza cuando se genera el código fuente.

Conforme nos movemos por los diferentes niveles de abstracción, trabajamos para crear abstracciones de datos y de procedimientos.

- Una **abstracción de datos** es un conjunto de datos que describen un objeto, como puede ser el DNI de una persona, que está compuesta por conjunto de partes de información, pero que nos podemos referir a todos los datos mencionando el nombre de la abstracción de datos.
- Una **abstracción procedimental** es una determinada secuencia de instrucciones que tienen una función limitada y específica, como puede ser “mover objeto”, que supone la secuencia de pasos “abrir pinza”, “mover hasta posición de destino 1”, “cerrar pinza”, “mover hasta posición 2”, “abrir pinza”, “mover hasta posición origen”, “cerrar pinza”.

Estas abstracciones permiten al diseñador representar un objeto a diferentes niveles de detalle.

3.2. REFINAMIENTO

El **refinamiento sucesivo** es una primera estrategia de diseño descendente propuesta por Niklaus Wirth. La arquitectura de un programa se desarrolla en niveles sucesivos de refinamiento de los detalles procedimentales. Se desarrolla una jerarquía descomponiendo una función de forma sucesiva hasta que se llega a las sentencias del lenguaje de programación.

Comenzamos con una declaración de la función (o una descripción de la información) definida a un nivel superior de abstracción. Es decir, la declaración describe la función o la información conceptualmente, pero no proporciona información sobre el funcionamiento interno de la función o sobre la estructura interna de la información, sino que se va a realizando sucesivamente, dando cada vez más detalles.

3.3. MODULARIDAD

El software se divide en componentes con nombres y ubicaciones determinados, que se denominan módulos y que se integran para satisfacer los requisitos del proveedor.

El software monolítico (es decir, un programa grande compuesto de un solo módulo) no puede ser estudiado fácilmente por un lector, ya que el número de caminos de control, el número de variables y la complejidad global harían el código prácticamente indescifrable.

Mátemáticamente, esto se explica de esta forma:

Sea *C(x)* una función que defina la complejidad de un problema *x*, y *E(x)* una función que defina el esfuerzo de desarrollo de un problema *x*.

Para dos problemas p_1 y p_2, si

$$C(p_1) > C(p_2)$$

se deduce que

$$E(p_1) > E(p_2)$$

Además, se cumple que

$$C(p_1 + p_2) > C(p_1) + C(p_2)$$

y que

$$E(p_1 + p_2) > E(p_1) + E(p_2)$$

Esto nos lleva a la conclusión **divide y vencerás**, por tanto la modularidad del software facilita el desarrollo del mismo, pero hasta un cierto límite, porque si llegáramos a dividir el problema en infinitos módulos, los módulos tendrían una complejidad y un esfuerzo mucho menor, pero crecería el coste asociado a la creación de interfaces entre los módulos, tal y como muestra la figura 3.

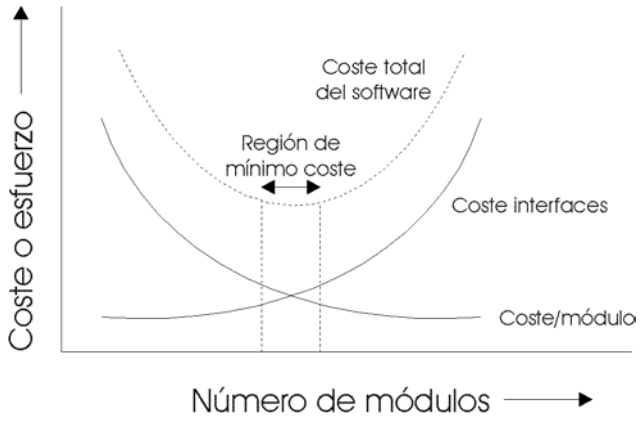

Figura 3. Modularidad y coste del software

3.4. ARQUITECTURA DEL SOFTWARE

La arquitectura del software se refiere a dos características importantes del software:

- La estructura jerárquica de los módulos del software
- La estructura de los datos

La arquitectura del software se obtiene mediante un proceso de partición, que relaciona los problemas del mundo real (definidos en el análisis de requerimientos) con las soluciones software para resolver los problemas software.

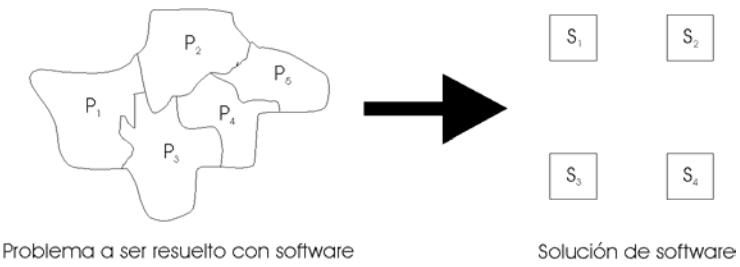

Figura 4. Evolución de la estructura

Este proceso de transición entre el análisis de requerimientos y el diseño se representa es el que representa la figura 4.

3.5. JERARQUÍA DE CONTROL

También se le conoce como **estructura del programa**, y representa la organización jerárquica de los módulos de un programa e implica una jerarquía de control.

La representación de jerarquía se suele representar con diagramas de árbol, aunque también se pueden utilizar otros tipos de notaciones.

La figura 5 muestra un ejemplo de una estructura de un programa.

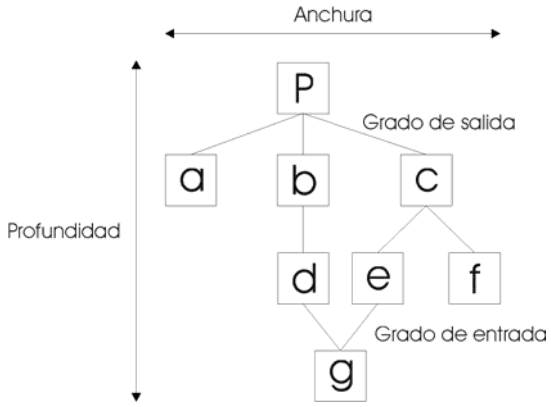

Figura 5. Estructura de un programa

A continuación definiremos los algunos términos relacionados con la figura 5.

- Profundidad: Número de niveles de control
- Anchura: Amplitud global del control
- Grado de salida: Número de módulos que controla un módulo
- Grado de entrada: Número de módulos que controlan a un módulo
- Visibilidad: Conjunto de componentes del programa que pueden ser invocados por un módulo (*Herencia en entornos de POO*). Todos los objetos serían visibles para el módulo
- Conectividad: Conjunto de componentes a los que se invoca directamente o se utilizan sus datos. (*La ejecución de un módulo puede suponer la ejecución de otro módulo*)

3.6. ESTRUCTURA DE DATOS

La estructura de datos es una representación de la lógica que existe entre los elementos individuales de información. Debido a que la estructura de la información afectará de forma determinante al diseño procedimiental, la estructura de datos es tan importante como la estructura del programa en la representación de la arquitectura del software.

La estructura de datos dicta la organización, los métodos de acceso, el grado de asociatividad y las alternativas para el tratamiento de la información.

Las estructuras de datos clásicas son los elementos escalares, los arrays, las listas y los árboles.

3.7. PROCEDIMIENTOS DEL SOFTWARE

La estructura del programa define la jerarquía de control, independientemente de las decisiones y secuencias de procesamiento. El procedimiento del software se centra en los detalles de procesamiento de cada módulo individual, como muestra la figura 6.

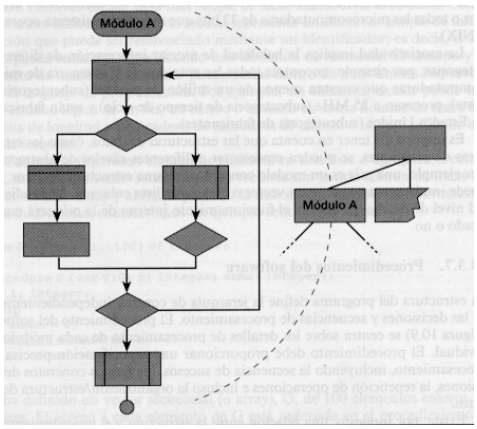

Figura 6. Procedimiento dentro de un módulo

El procedimiento debe proporcionar una especificación precisa del procesamiento, incluyendo la secuencia de sucesos, los puntos concretos de decisiones, la repetición de operaciones e incluso la organización/estructura de los datos.

Como existe una relación entre la estructura y el procedimiento, ya que el procesamiento de un módulo puede suponer la llamada a otros módulos. A esto se le conoce como **representación procedimental del software por capas**, como muestra la figura 7

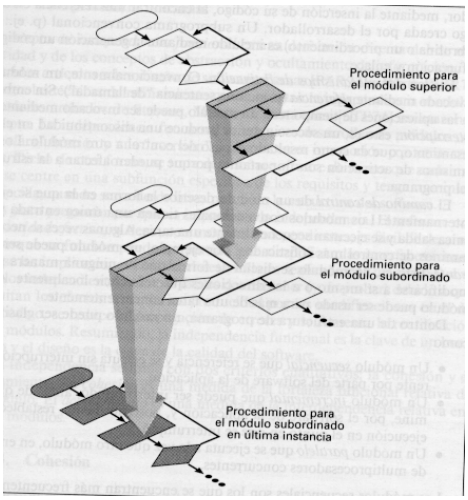

Figura 7. Procedimiento realizado por capas

3.8. OCULTAMIENTO DE INFORMACIÓN

El concepto de modularidad nos lleva a esta pregunta: ¿cómo descomponer una solución de software en el mejor conjunto de módulos?

El principio de ocultamiento de la información sugiere que los módulos deben especificarse de forma que la información (procedimientos y datos) contenida dentro de un módulo sea inaccesible a otros módulos que no necesiten tal información.

Por tanto se trata de definir una serie de módulos independientes que se comuniquen sólo a través de la información necesaria para realizar la función de software.

El uso de ocultamiento de información en el diseño facilitará las modificaciones, prueba y mantenimiento del software, ya que como la mayoría de los datos y de los procedimientos están ocultos a otras partes del software, será menos probable que los errores que se introduzcan durante la modificación se propaguen a otros módulos del software.

4. Diseño modular efectivo

Todos los fundamentos del diseño anteriores sirven para incentivar los diseños modulares.

Un diseño modular:

- Reduce la complejidad
- Facilita los cambios
- Implementación más sencilla
- Permite el desarrollo paralelo de partes diferentes de un sistema

4.1. TIPOS DE MÓDULOS

Para la definición de módulos en una arquitectura de software se utiliza la abstracción y ocultamiento de información. Estos atributos tienen que ser traducidos a las características de ejecución del módulo, caracterizadas por el historial de ejecución, el mecanismo de activación y el camino de control, y que describimos a continuación:

- El *historial de incorporación* se refiere al momento en que se incluye el módulo en la descripción del software en lenguaje fuente.
- El *mecanismo de activación* se refiere a la forma en que se invoca a un módulo, que puede ser de **referencia** (mediante una llamada) o de **interrupción** (en aplicaciones en tiempo real, ocurre un evento en el mundo exterior)
- El *camino de control* de un módulo describe la forma en que se ejecuta internamente, y son los que se describen a continuación.

4.1.1. Módulos secuenciales

Se ejecutan sin interrupción aparente por parte del software de la aplicación, es decir ejecutan secuencialmente una tarea.

4.1.2. Módulos incrementales

También se les conoce como **corrutinas**, y pueden ser interrumpidos antes de que terminen por el software de la aplicación, y restablecerse posteriormente su ejecución en el punto en que se interrumpió.

4.1.3. Módulos paralelos

Un módulo paralelo se ejecuta a la vez que otro módulo en entornos multiprocesadores.

4.2. INDEPENDENCIA FUNCIONAL

La independencia funcional es una derivación directa de la modularidad, de la abstracción y del ocultamiento de información.

La independencia funcional se adquiere desarrollando módulos con una función clara y con pocas relaciones con otros módulos, de forma que cada módulo se centra en una subfunción específica de los requerimientos y tenga una interfaz sencilla.

Esta independencia tiene varias consecuencias positivas como son:

- Módulos independientes fáciles de desarrollar
- Creación de interfaces sencillas
- Facilidad para la prueba y el mantenimiento
- Se reduce la propagación de errores
- Se fomenta la reutilización de módulos.

La independencia se mide con dos criterios cualitativos que son la cohesión y el acoplamiento.

4.2.1. Cohesión

La cohesión es una extensión del concepto de ocultamiento de información. Un modulo cohesivo ejecuta una tarea sencilla de un procedimiento de software y requiere poca interacción con procedimientos que ejecutan otras partes de un programa.

Dicho de forma sencilla, un módulo cohesivo sólo hace (idealmente) una cosa.

Por tanto el diseñador debe comprender lo que es la cohesión y evitar la baja cohesión en el diseño de los módulos.

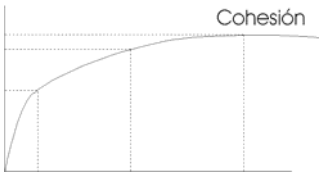

Figura 8. Representación de la escala de cohesión

Lo importante es intentar conseguir una cohesión alta y saber reconocer la cohesión baja, de forma que pueda modificar el diseño del software para tener de una mayor independencia funcional.

4.2.2 Acoplamiento

El acoplamiento es una medida de la interconexión entre los módulos de una estructura de programa.

El acoplamiento depende de la complejidad de las interfaces entre los módulos y de los datos que pasan a través de la interfaz.

En el diseño de software buscamos el acoplamiento más bajo posible. Una conectividad sencilla entre módulos da como resultado un software más fácil de comprender y menos propenso al *efecto onda* (propagación de errores a lo largo del sistema).

5. Diseño de datos

El diseño de datos es la primera de las tres actividades de diseño realizadas durante la ingeniería del software. El impacto de la estructura de datos sobre la estructura de programa y la complejidad procedimental, hace que el diseño de datos tenga una gran influencia en la calidad del software.

Los conceptos de ocultamiento de información y de abstracción de datos conforman la base de los métodos de diseño de datos.

Según Wasserman "*La actividad principal durante la fase de diseño de datos es la selección de las representaciones lógicas de las estructuras de datos, identificados durante las fases de definición y especificación de requerimientos. Una actividad importante durante el diseño es la de identificar los módulos de programa que deben operar directamente sobre las estructuras de datos. De esta forma, puede restringirse el ámbito del efecto de las decisiones concretas de diseño de datos.*"

Los datos bien diseñados conducen a:

- Mejor estructura de programa
- Modularidad efectiva
- Complejidad procedimental reducida

5.1. PRINCIPIOS PARA LA ESPECIFICACIÓN DE DATOS

Los principios que se citan a continuación son la base del método de diseño de datos, y una definición clara de la información es esencial para el desarrollo de un buen software.

1. *Los principios sistemáticos de análisis aplicados a la función y el comportamiento también deben aplicarse a los datos.*

Al igual que en la fases de análisis del sistema, también se debe:

- Desarrollar y revisar las representaciones del flujo y del contenido de los datos.
- Identificar las estructuras de datos.
- Considerar estructuras de datos alternativas.
- Evaluar el impacto de la modelización de datos sobre el diseño del software

Por ejemplo, la especificación de una lista enlazada circular puede satisfacer los requerimientos de los datos, pero también puede conducir a un diseño del software difícil de manejar, por lo que deberemos ver si existen otras estructuras de datos alternativas que lleven a resultados mejores.

2. *Se deben identificar todas las estructuras de datos y las operaciones que se han de realizar sobre cada una de ellas.*

El diseño de una estructura eficiente debe tener en cuenta las operaciones que han de realizarse sobre dicha estructura de datos.

Por ejemplo, la especificación de un tipo abstracto de datos puede simplificar considerablemente el diseño del software.

3. *Debe establecerse y usarse un diccionario de datos para definir el diseño de los datos y del programa.*

Un diccionario de datos representa explícitamente las relaciones entre los datos y las restricciones sobre los elementos de una estructura de datos.

4. *Se deben posponer las decisiones de diseño de datos de bajo nivel hasta más adelante en el proceso de datos.*

Para el diseño de datos puede seguirse un proceso de refinamiento sucesivo, es decir, puede definirse una organización global de los datos durante el análisis de requerimientos, refinarse durante el diseño preliminar y especificarse en detalle en el diseño detallado.

5. *La representación de una estructura de datos sólo debe ser conocida por los módulos que hagan uso directo de los datos contenidos en la estructura.*

El concepto de ocultamiento de información y el concepto de acoplamiento asociado proporcionan una valoración importante de la calidad del diseño del software.

6. *Se debe desarrollar una librería de estructuras de datos útiles y de las operaciones que se le pueden aplicar.*

Se pueden diseñar las estructuras de datos de forma que sean reusables, de forma que las estructuras de datos y las operaciones se vean como recursos para el diseño de software.

7. *El diseño de software y el lenguaje de programación deben soportar la especificación y realización de tipos abstractos de datos*.

La implementación de una estructura de datos compleja puede convertirse en una tarea muy difícil si no hay una forma directa de realizar una especificación directa de la estructura de datos.

6. Diseño arquitectónico

El objetivo principal del diseño arquitectónico es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos.

Además el diseño arquitectónico mezcla la estructura de programas y la estructura de datos y define las interfaces que facilitan el flujo de los datos a lo largo del programa.

Se trata de no centrarse en los detalles y código de los procedimientos (tareas que se realizan más adelante) sino en centrarse en la arquitectura del software que permita obtener una visión general del software.

7. Diseño procedimental

EL diseño procedimental se realiza después de que se ha establecido la estructura del programa y de los datos. La especificación procedimental que define los algoritmos, cabe pensar que se podría especificar en lenguaje natural, pero debido a la cantidad de ambigüedades que este lenguaje acarrea, es necesario utilizar una forma más restringida de representación.

7.1. PROGRAMACIÓN ESTRUCTURADA

A finales de los 60 se propuso la utilización de un conjunto de construcciones lógicas con las que podía formarse cualquier programa.

Cada construcción tenía estructura lógica predecible. Se entra por ella por el principio, se sale por el final y facilita al lector el seguimiento del flujo procedimental.

Las construcciones son *secuencia, condición* y *repetición*, y son fundamentales en la programación estructurada.

- La secuencia implementa los pasos de procedimiento esenciales de la especificación de cualquier algoritmo.
- La condición da la posibilidad de seleccionar un procedimiento basándose en alguna ocurrencia lógica.
- La repetición proporciona iteración.

Las construcciones estructuradas se propusieron para limitar el diseño procedimental del software a un número pequeño de operaciones predecibles, lo que facilita la legibilidad, prueba y mantenimiento del software.

Además, la notación debe facilitar la codificación, de forma que el código se obtenga de forma natural a partir del diseño.

7.1.1. Notaciones para la representación gráfica en diseño procedimental

Para evitar desarrollar un software erróneo, es fundamental que se utilicen correctamente las herramientas gráficas para el diseño, como son los diagramas de flujo y los diagramas de cajas.

Diagrama de flujo

Es la representación gráfica que más se utiliza en el diseño procedimental.

Para representar un paso de procesamiento se utiliza un cuadro, para representar una condición se utiliza un rombo, y para representar el flujo de control se utilizan flechas.

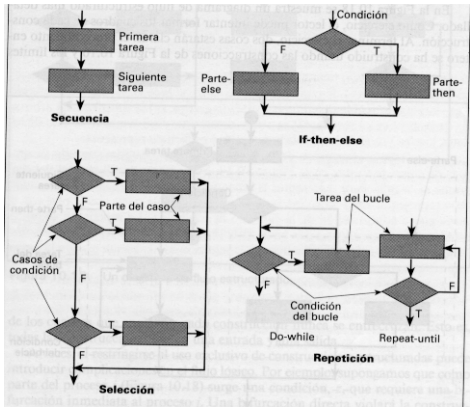

Figura 9. Construcciones de programación estructurada con diagramas de flujo

Las construcciones estructuradas pueden estar anidadas unas dentro de otras, cada una de estas puede realizar llamadas a módulos, teniendo entonces una disposición por **capas procedimentales** en un diagrama de flujo estructurado, como muestra la figura 10.

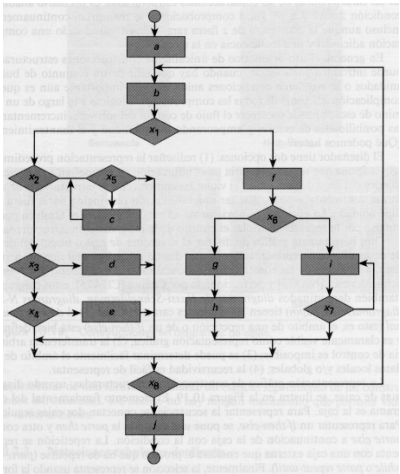

Figura 10. Un diagrama de flujo estructurado

A veces, el restringirse al uso exclusivo de construcciones estructuradas puede introducir complicaciones en el flujo lógico. Por ejemplo, supongamos que como parte del proceso *i* surge una condición, *z*, que requiere una bifurcación inmediata al proceso *j*. Una bifurcación directa violará la construcción lógica, escapando del dominio funcional de la sentencia *repeat-until*, de la cual forma parte el proceso *i*.

Para implementar la bifurcación anterior sin violar las características de las condiciones estructuradas, sería necesario añadir la condición *z* a *x7* y a *x8*. Estas comprobaciones se realizarían continuamente, incluso aunque no fuese necesario la ocurrencia de la *z*. De esta forma, habríamos introducido una condición adicional y una eficiencia en la ejecución. Entonces ¿qué podemos hacer?.

El diseñador tiene dos opciones:

- Rediseñar la representación procedimental, de forma que no sea necesaria la "bifurcación de escape" en una posición interior del flujo de control.
- Violar las construcciones estructuradas de forma controlada, es decir, diseñar una bifurcación restringida hacia fuera del flujo anidado.

Diagrama de cajas

Esta notación surgió del deseo de desarrollar una representación para el diseño procedimental que no permitiera la violación de construcciones estructuradas. Estos diagramas fueron desarrollados por Nassi y Schneiderman y perfeccionados por Chapin.y tienen las características siguientes:

- El ámbito funcional está bien definido y es claramente visible.
- La transferencia de control arbitraria es imposible.
- Es fácil determinar el ámbito de los datos locales y globales
- La recursividad es fácil de representar

La representación gráfica de construcciones estructuradas, usando diagramas de cajas, se muestra en la figura 11. El elemento fundamental del diagrama de cajas es la caja.

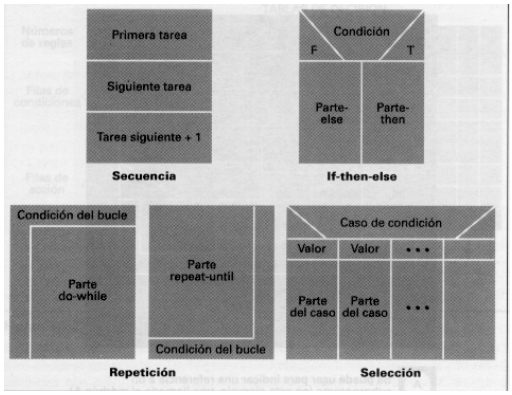

Figura 11. Construcciones en diagramas de cajas

Al igual que con los diagramas de flujo, se pueden crear diagramas de cajas por capas en múltiples páginas. Se puede representar una llamada a un módulo subordinado mediante una caja con el nombre del módulo encerrado dentro de una circunferencia.

La figura 12 muestra un diagrama de cajas que representa el mismo flujo de control que el de la figura 10.

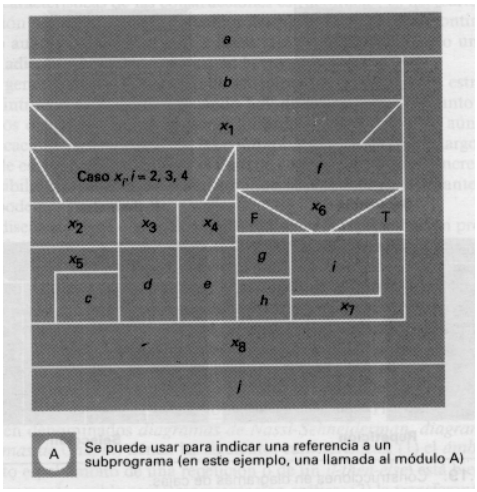

Figura 12. Un diagrama de cajas estructurado

Para ver la relativa facilidad con que puede comprenderse el dominio funcional, puede observarse el bucle *repeat-until* con la condición *x8*. Todas las construcciones lógicas contenidas dentro del bucle se encuentran fácilmente, debido a que están dispuestas en cajas interiores.

7.1.2. Notaciones tabulares de diseño

En muchas aplicaciones de software puede ser necesario que un módulo evalúe una compleja combinación de condiciones, y de acuerdo con ellas, seleccione la opción adecuada.

Las **tablas de decisión** constituyen una notación que traduce las acciones y condiciones a una forma tabular.

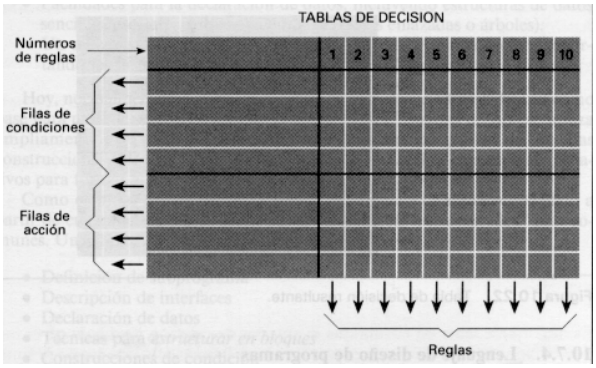

Figura 13. Esqueleto de una tabla de decisión

La figura 13 muestra la organización de una tabla de decisión, que está dividida en cuatro cuadrantes por las líneas gruesas.

- El cuadrante superior izquierdo contiene una lista de todas las condiciones
- El cuadrante inferior izquierdo contiene una lista de todas las acciones que se pueden realizar en función de las condiciones
- Los cuadrantes de la derecha forman una matriz que contienen las combinaciones de las condiciones para producir las acciones.

Por tanto, cada columna de la matriz representa una **regla de procesamiento**.

Los pasos para la creación de una tabla de decisión son los siguientes:

- Listar todas las acciones que se pueden asociar a un módulo.
- Listar todas las condiciones necesarias para la ejecución de un procedimiento.
- Asociar conjuntos de condiciones específicas a acciones específicas, eliminando combinaciones imposibles. Desarrollar las posibles combinaciones de condiciones.
- Definir reglas indicando las acciones que ocurren para una serie de condiciones.

7.1.3. Lenguaje de diseño de programas

Un lenguaje de diseño de programas (LDP), también conocido como lenguaje estructurado o pseudocódigo es un lenguaje que utiliza el vocabulario de un lenguaje y la sintaxis de otro.

Independientemente de su origen, un LDP tiene que tener las características siguientes:

- Una sintaxis fija de **palabras clave** que permitan construir todas las construcciones estructuradas, declarar datos y establecer características de modularidad.
- Una sintaxis libre en lenguaje natural para describir las características de procesamiento.
- Facilidades para la declaración de datos, incluyendo estructuras de datos simples y complejas.
- Un mecanismo de definición de subprogramas y de llamada a éstos.

8. Documentación del diseño

Se puede utilizar un esquema de documento como el siguiente para la especificación del diseño.

Especificación de diseño

1. Ambito
   - 1.1 Objetivos del sistema
   - 1.2 Hardware, software e interfaces humanas
   - 1.3 Principales funciones del software
   - 1.4 Base de datos definida externamente
   - 1.5 Principales restricciones y limitaciones del diseño
2. Documentos de referencia
   - 2.1 Documentación del software existente
   - 2.2 Documentación del sistema
   - 2.3 Documentos del vendedor (hardware o software)
   - 2.4 Referencia técnica
3. Descripción del diseño
   - 3.1 Descripción de datos
     - 3.1.1 Revisión del flujo de datos
     - 3.1.2 Revisión de la estructura de datos
   - 3.2 Estructura de programa derivada
   - 3.3 Interfaces dentro de la estructura
4. Módulos (Para cada módulo)
   - 4.1 Texto explicativo
   - 4.2 Descripción de la interfaz
   - 4.3 Descripción en lenguaje de diseño
   - 4.4 Módulos utilizados
   - 4.5 Organización de los datos
   - 4.6 Comentarios
5. Estructuras de archivos y datos globales
   - 5.1 Estructura de archivos internos
     - 5.1.1 Estructura lógica
     - 5.1.2 Descripción lógica de los registros
     - 5.1.3 Método de acceso
   - 5.2 Datos globales
   - 5.3 Referencias cruzadas entre archivos y datos (ver figura 14)
6. Referencias cruzadas para los requisitos
7. Provisiones de prueba
   - 7.1 Directrices de prueba
   - 7.2 Estrategia de integración
   - 7.3 Consideraciones especiales
8. Empaquetamiento
   - 8.1 Provisiones especiales de solapamiento del programa
   - 8.2 Consideraciones de transferencia
9. Notas especiales
10. Apéndices

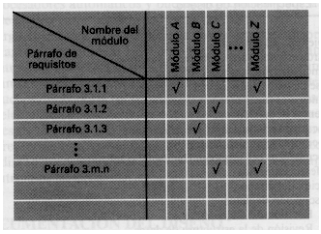

Figura 14. Referencias cruzadas entre archivos y datos

En la sección 1 se describe el ámbito global del trabajo de diseño. Gran parte de la información de esta sección se deriva de la especificación del sistema y de otros documentos obtenidos en la fase de definición del software.

La sección 2 incluye las referencias específicas a la documentación de soporte.

En la sección 3, que se desarrolla durante el diseño preliminar, se refinan y utilizan los DFD y otras representaciones de los datos, desarrollados durante el análisis de requerimientos. Puesto que está definido el flujo de la información se pueden desarrollar las descripciones de la interfaz para los elementos del software.

Las secciones 4 y 5 se realizan durante el paso del diseño preliminar al diseño detallado. Inicialmente se describen los módulos mediante una descripción procedimental del módulo en lenguaje natural. Después se utiliza una herramienta de diseño procedimental para traducir el texto explicativo a una descripción estructurada.

La sección 5 contiene una descripción de la organización de los datos, se asignan los datos globales y se establecen referencias cruzadas que conectan los módulos individuales con los archivos o datos globales.

La sección 6 contiene las referencias cruzadas entre los requerimientos y los módulos que los implementan y facilita la identificación de los módulos que se corresponden con requerimientos críticos.

La sección 7 contiene la primera etapa del desarrollo del procedimiento de prueba. Una vez que se han establecido la estructura y las interfaces del software podemos desarrollar directrices para la prueba de los módulos individuales y de su integración.

La sección 8 contiene las restricciones del diseño, tales como las limitaciones físicas de memoria o la necesidad de un gran rendimiento del software, y también describe el método que se usará para transferir el software al lugar del cliente.

Las secciones 9 y 10 contienen datos complementarios como las descripciones de algoritmos o procedimientos alternativos y otro tipo de información relevante.