Capítulo

4

Fundamentos del Diseño de Software

Centro Asociado Palma de Mallorca

Tutor: Antonio Rivero Cuesta

4 Fundamentos del Diseño de Software

4.1 Introducción.

En este tema se inicia el estudio de las etapas de desarrollo. Después de haber especificado QUE se quiere resolver durante la especificación, las etapas de desarrollo se dedican a determinar COMO se debe resolver el proyecto.

La primera de estas etapas es la de diseño, se continua con la de codificación y se finaliza con las etapas de pruebas del software realizado. En este tema se abordan los fundamentos de la etapa de diseño.

Diseño de software: consiste en definir y formalizar la estructura del sistema con el suficiente detalle como para permitir su realización física.

El punto de partida principal para abordar el diseño es el documento de especificación de requisitos (SRD).

En la realización del diseño podemos destacar algunas características:

Es un proceso creativo que no es nada trivial.

Casi siempre se lleva a cabo de una forma iterativa mediante prueba y error.

Es muy importante la experiencia previa. El método más eficaz es participar en algún diseño y aprender de otros diseñadores sus técnicas de trabajo. También es aconsejable estudiar diseños ya realizados y analizar las razones por las que se adopta una u otra solución.

Se tratara de reutilizar el mayor número de módulos o elementos ya desarrollados. Si no es posible por tratarse de un diseño completamente original , al menos se podrá aprovechar el enfoque dado a algún otro proyecto anterior, lo que se conoce como aprovechar el KNOW-HOW (Saber hacer).

4.2 Objetivos

Qué entendemos por diseño y analizar las actividades que se deben realizar para llevarlo a cabo.

Conocer conceptos fundamentales a tener en cuenta para realizar cualquier diseño.

Conocer distintas notaciones.

Conocer los documentos de diseño arquitectónico y del diseño detallado.

4.3 ¿Qué es el diseño?

Objetivo fundamental del diseño: Es conseguir que el software sea fácil de mantener, y si es posible reutilizarlo en futuras aplicaciones. Para conseguir ambos objetivos, la etapa de diseño es la más importante de la fase de desarrollo de software.

Durante el diseño se tiene que pasar gradualmente de las ideas informales recogidas en el SRD a definiciones detalladas y precisas para la realización del software mediante refinamientos sucesivos.

Actividades habituales en el diseño de un Sistema:

Diseño arquitectónico: Se abordan aspectos estructurales y de organización del sistema y su posible subdivisión en subsistemas o módulos. Además se tienen que establecer las relaciones entre los subsistemas creados y definir las interfaces entre ellos.

Diseño detallado: Se aborda la estructura de cada uno de los módulos en los que quedó subdividido el sistema global. Si por ejemplo utilizáramos Modula-2 como notación de diseño, en esta actividad elaboraríamos los módulos de definición para cada uno de los módulos del programa global.

Diseño procedimental: Se abordan la organización de las operaciones o servicios que ofrecerá cada uno de los módulos. Siguiendo con el ejemplo de Modula-2, en esta actividad se diseñan (en pseudocódigo) los algoritmos más importantes que se emplean en los módulos de implementación.

Diseño de datos: Se aborda la organización de la base de datos del sistema. Se parte del diccionario de datos y de los diagramas E - R de la especificación del sistema. Con esta actividad se trata de concretar el formato exacto para cada dato y la organización que debe existir entre ellos. El diseño de datos es muy importante para conseguir que el sistema sea utilizable y fácilmente mantenible.

Diseño de la interfaz de usuario: Aborda la organización de la interfaz del usuario, para conseguir un diseño más ergonómico. Es decir, se intenta conseguir un dialogo mas ergonómico entre el usuario y el computador.

El resultado de todas estas actividades de diseño es el **Documento de Diseño de Software (SDD)** que contendrá una especificación lo más formal posible de la estructura global del sistema y de cada uno de los elementos del mismo.

Si la complejidad del sistema así lo aconseja se podrán utilizar varios documentos para describir de forma jerarquizada la estructura global del sistema, la estructura de los elementos de un primer nivel de detalle y en los sucesivos niveles de detalle hasta alcanzar el nivel de codificación con los listados de los programas.

Según las normas ESA tenemos 2 documentos de diseño:

- **ADD:** Documento de Diseño Arquitectónico.
- **DDD:** Documento de Diseño Detallado, al que se le puede añadir como apéndices los listados de los programas una vez completado el desarrollo.

4.4 Conceptos de base

A continuación se listan los conceptos más interesantes a tener en cuenta en cualquier diseño:

4.4.1 Abstracción

Cuando se diseña un nuevo sistema software es importante identificar los elementos significativos de los que consta y abstraer la utilidad específica de cada uno, incluso más allá del sistema software para el que se está diseñando. Gracias a esto el elemento diseñado podrá ser sustituido en el futuro por otro con mejores prestaciones y también podrá ser reutilizado en otros proyectos similares. Por tanto, se cumplen los dos objetivos principales del diseño que son, conseguir elementos fácilmente mantenibles y reutilizables.

Durante el proceso de diseño se debe aplicar el concepto de abstracción en todos los niveles de diseño. Por ejemplo, para el sistema de control de acceso del tema anterior tenemos:

En un primer nivel aparecen abstracciones tales como Tarjeta, Mensajes, Órdenes, etc. Inicialmente todos ellos son elementos abstractos y su diseño se puede realizar sin tener en cuenta al Sistema de Control de Acceso concreto en el que se utilizaran. Con esto conseguimos:

- Facilitar los cambios futuros.
- Posibilidad de reutilizarlo en diversos sistemas.

En un segundo nivel aparecen nuevas abstracciones como Clave, Control de Puerta, Comprobar Clave, etc. a los cuales se aplicaran los mismos criterios.

Este proceso de abstracción se debe continuar con cada elemento software que tengamos que diseñar.

En el diseño de los elementos software se pueden utilizar fundamentalmente tres formas de abstracción:

Abstracciones funcionales: sirven para crear expresiones o acciones parametrizadas mediante el empleo de funciones o procedimientos. Para diseñar una abstracción funcional es necesario fijar:

Los parámetros o argumentos que se le deben pasar.

El resultado que devolverá en el caso de una expresión parametrizada.

Lo que se pretende que resuelva.

Como lo debe resolver, es decir, el algoritmo que debe emplear.

Por ejemplo, se puede diseñar una abstracción funcional para "Comprobar Clave" cuyo parámetro es la clave a comprobar y que devuelva como resultado SI/NO la clave es correcta.

Tipos abstractos: sirven para crear los nuevos tipos de datos que se necesitan para abordar el diseño del sistema. Por ejemplo, un tipo Tarjeta para guardar toda la información relacionada con la tarjeta utilizada: clase de tarjeta, identificador, clave de acceso, datos personales, etc.

Además de esto, junto al nuevo tipo de dato se deben diseñar todos los métodos u operaciones que se pueden realizar con él. Por ejemplo, Leer Tarjeta, Grabar Tarjeta, Comprobar Tarjeta, etc.

Máquinas abstractas: permiten establecer un nivel de abstracción superior a los dos anteriores y en él se define de una manera formal el comportamiento de una maquina. Un ejemplo de este tipo de abstracción seria un intérprete de SQL para la gestión de una base de datos. El lenguaje SQL establece formalmente el comportamiento perfectamente definido para la maquina abstracta encargada de manejar la base de datos: construcción, búsqueda e inserción de elementos en la base de datos. La tarea de diseño consiste en este caso en definir formalmente la máquina abstracta que se necesita.

4.4.2 Modularidad

Uno de los primeros pasos que se debe dar al abordar un diseño es dividir el sistema en sus correspondientes módulos o partes claramente diferenciadas. Esta división permite encargar a personas diferentes el desarrollo de cada módulo y que todas ellas puedan trabajar simultáneamente. Podemos citar como ventajas de utilizar un diseño modular las siguientes:

Claridad: siempre es más fácil entender y manejar cada una de las partes de un sistema que tratar de entenderlo como un todo compacto.

Reducción de Costos: resulta más barato desarrollar, depurar, documentar, probar y mantener un sistema modular que otro que no lo es, excepto si el número de módulos crece innecesariamente.

Reutilización: si los módulos se diseñan teniendo en cuenta otras posibles aplicaciones resultara inmediata su reutilización.

La modularidad es un concepto de diseño que no debe estar ligado a la etapa de codificación y mucho menos al empleo de un determinado lenguaje de programación.

4.5 Refinamiento

En un diseño siempre se parte inicialmente de una idea no muy concreta que se va refinando en sucesivas aproximaciones hasta perfilar el más mínimo detalle.

El objetivo global de un nuevo sistema software expresado en su especificación se debe refinar en sucesivos pasos hasta que todo quede expresado en el lenguaje de programación del computador. Es decir, a través del refinamiento pasaremos de la especificación del SRD al lenguaje de programación del computador.

El proceso de refinamiento se puede dar por concluido cuando todas las acciones y expresiones quedan refinadas en función de otras acciones o expresiones o bien en función de las instrucciones básicas del lenguaje empleado.

4.5.1 Estructuras de datos

Para el diseño deberemos tener en cuenta las estructuras fundamentales que son: Registros, Conjuntos, Formaciones, Listas, Pilas, Colas, Árboles, Grafos, Tablas, Ficheros.

Es labor del diseñador la búsqueda, a partir de estas estructuras básicas, de la combinación más adecuada para lograr aquella estructura o estructuras que den respuesta a las necesidades del sistema planteadas en su especificación.

4.5.2 Ocultación

Para poder utilizar correctamente un programa no es necesario conocer cuál es su estructura interna. Por tanto, al programador "usuario" de un módulo desarrollado por otro programador del equipo puede

quedarle completamente oculta la organización de los datos internos que maneja y el detalle de los algoritmos que emplea.

Cuando se diseña la estructura de cada uno de los módulos de un sistema, se debe hacer de tal manera que dentro de él queden ocultos todos los detalles que resultan irrelevantes para su utilización.

Con carácter general, se trata de ocultar al usuario todo lo que pueda ser susceptible de cambio en el futuro y además es irrelevante para el uso. Sin embargo, se muestra en la interfaz solo aquello que resultará invariable con cualquier cambio. Las ventajas de aplicar este concepto son las siguientes:

Depuración: resulta más sencillo detectar que módulo concreto no funciona correctamente. También es más fácil establecer estrategias y programas de prueba que verifiquen y depuren cada módulo por separado en base a lo que hacen sin tener en cuenta como lo hacen.

Mantenimiento: cualquier modificación u operación de mantenimiento que se necesite en un módulo concreto no afectara al resto de los módulos del sistema.

4.5.3 Genericidad

En este caso, un posible enfoque de diseño es, agrupar aquellos elementos del sistema que utilizan estructuras semejantes o que necesitan de un tratamiento similar. Posteriormente, cada uno de los elementos agrupados se puede diseñar como un caso particular del elemento genérico.

Por ejemplo, supongamos que tenemos que diseñar un sistema operativo multitarea que necesita atender las órdenes que llegan de distintos terminales. Las órdenes habituales son:

- Imprimir por cualquiera de las impresoras disponibles.
- Acceder desde los terminales a ficheros y bases de datos compartidas.

Cada una de estas actividades necesita de un módulo gestor para decidir en qué orden se atienden las peticiones. La forma más sencilla de gestión es poner en cola las órdenes, peticiones de impresión o peticiones de acceso a ficheros o bases de datos compartidas. Como vemos, surge la necesidad de una estructura genérica en forma de cola para la que se necesitan también unas operaciones genéricas tales como poner en cola, sacar el primero, etc. Ahora, en cada caso el tipo de información que se guarda en la cola es diferente: tipo de orden a ejecutar, texto a imprimir, etc. A partir de la cola genérica y con un diseño posterior más detallado se tendrá que decidir la estructura concreta de las distintas colas necesarias y si para alguna de ellas es conveniente utilizar prioridades, lo que daría lugar a operaciones especificas.

A pesar de todo, tenemos que tener en cuenta que en algunos lenguajes de programación el concepto de genericidad se puede ver desvirtuado. Por ejemplo, Modula-2 impone restricciones muy fuertes a la hora de manejar datos de distinto tipo y las operaciones entre ellos. En este caso sería necesario definir un tipo distinto de cola para cada tipo de elemento a almacenar, y aunque las operaciones sean esencialmente idénticas para los distintos datos almacenados, también es necesario implementar como operaciones distintas las destinadas a manejar cada tipo de cola. Si tuviéramos, por ejemplo los tipos:

- Cola_de_enteros.
- Cola_de_reales.

Sería necesario implementar las operaciones:

- Poner_entero (en la cola de enteros).
- Poner _real (en la cola de reales).

Como vemos, esta solución que es la única posible en estos lenguajes, desvirtúa de forma evidente la genericidad propuesta en el diseño. En estos casos convendría utilizar lenguajes que tengan la posibilidad de declarar elementos genéricos como por ejemplo ADA.

4.5.4 Herencia

Otro enfoque posible del diseño cuando hay elementos con características comunes es establecer una clasificación o jerarquía entre esos elementos del sistema partiendo de un elemento "padre" que posee una estructura y operaciones básicas. Los elementos "hijos" heredan del "padre" su estructura y operaciones para ampliarlos, mejorarlos o simplemente adaptarlos a sus necesidades. A su vez los elementos "hijo" pueden tener otros "hijos" que hereden de ellos de una forma semejante. De manera consecutiva se puede continuar con los siguientes descendientes hasta donde sea necesario. Esta es la idea fundamental en la que se basa el concepto de herencia.

El mecanismo de herencia tiene como objetivo fundamental facilitar la reutilización de software ya desarrollado.

Por ejemplo, supongamos que tratamos de realizar un software para dibujo asistido por computador. Las figuras que se manejan se podrían clasificar del siguiente modo

FIGURAS:

ABIERTAS:

TRAZO RECTO:

LINEA RECTA.

TRAZO CURVO:

SEGMENTO DE CIRCUNFERENCIA.

CERRADAS:

ELIPSES:

CIRCULOS.

POLIGONOS:

TRIANGULOS:

EQUILATEROS.

RECTANGULOS:

CUADRADOS.

En este ejemplo, el elemento "padre" será FIGURAS. Las operaciones básicas con cualquier tipo de figura podrán ser:

- Desplazar.
- Rotar.
- Pintar.

Tendríamos que tener en cuenta dos puntos:

Aunque estas operaciones las heredan todos los tipos de figura, deberán ser adaptadas en cada caso. Así, Rotar los CIRCULOS significa dejarlos tal cual están.

Al concretar los elementos "hijos" aparecen nuevas operaciones que no tenían sentido en el elemento "padre". Así, la operación de calcular el Perímetro solo tiene sentido para figuras CERRADAS., etc.

El concepto de herencia está muy ligado a las metodologías de análisis y diseño de software orientado a objetos. Además, es posible realizar diseños en los que se tengan en cuenta herencias múltiples de varios "padres".

4.5.5 Polimorfismo

El concepto de polimorfismo engloba distintas posibilidades utilizadas habitualmente para conseguir que un mismo elemento software adquiera varias formas simultáneamente:

El ***concepto de genericidad*** es una manera de lograr que un elemento genérico pueda adquirir distintas formas cuando se particulariza su utilización.

El ***concepto de Polimorfismo*** está muy unido al concepto de herencia. Hay tres tipos:

De Anulación: las estructuras y operaciones heredadas se pueden adaptar a las necesidades concretas del elemento "hijo". En el ejemplo de antes no sería igual Rotar Elipses (Padre) que Círculos (hijo). Por tanto, la operación de rotar tiene distintas formas según el tipo de figura a la que se destina y es en el momento de la ejecución del programa cuando se utiliza una u otra forma de rotación. Este tipo de Polimorfismo se conoce como de Anulación dado que la rotación específica para los círculos anula la más general para las elipses.

Diferido: Cada uno de los hijos concreta la forma específica de la operación del padre. Por ejemplo, la operación Rotar FIGURAS resultara muy compleja y probablemente inútil.

Sobrecarga: Ahora son los operadores, funciones y procedimientos los que toman múltiples formas. Por ejemplo, los operadores +,-,*,/ son similares para operaciones entre enteros , reales , conjuntos o matrices. Sin embargo en cada caso el tipo de operación que se invoca es distinto:

La suma entre enteros es mucho más sencilla y rápida que la suma entre reales.

Con el mismo operador + se indica la operación suma entre valores numéricos o la operación de unión entre dos conjuntos o la suma de matrices.

Este mismo concepto (Sobrecarga) se debe aplicar cuando se diseñan las operaciones a realizar entre elementos del sistema que se trata de desarrollar.

Por ejemplo, se puede utilizar el operador + para unir ristras de caracteres o realizar resúmenes de ventas.

También se puede utilizar este tipo de polimorfismo con funciones y procedimientos. Por ejemplo, podemos tener la función Pintar para FIGURAS y la función Pintar para representar en una grafica los valores de una tabla.

Todas estas posibilidades de Polimorfismo redundan en una mayor facilidad para realizar software reutilizable y mantenible.

4.5.6 Concurrencia

Se aprovecha la capacidad de proceso del computador, ejecutando tareas de forma concurrente. Si se diseña un sistema con restricciones de tiempo hay que tener en cuenta lo siguiente:

Tareas concurrentes: ver que tareas se deben ejecutar en paralelo para respetar las restricciones impuestas. Se deberá prestar especial atención a aquellas tareas con tiempos de respuesta más críticas y aquellas otras que se ejecutan con mayor frecuencia.

Sincronización de tareas: determinar los puntos de sincronización entre las distintas tareas con semáforos o monitores.

Comunicación entre las tareas: distinguir si la cooperación se basa en datos compartidos o en paso de mensajes entre las tareas. Si se utilizan datos compartidos se tendrá que evitar que los datos puedan ser modificados en el momento de la consulta. En este caso es necesario utilizar mecanismos como semáforos, monitores, regiones críticas, etc.

Para garantizar la exclusión mutua entre las distintas tareas que modifican y consultan los datos compartidos.

Interbloqueo**:** estudiar los posibles interbloqueos entre tareas. Un interbloqueo se produce cuando una o varias tareas permanecen esperando por tiempo indefinido una situación que no se puede producir nunca.

El concepto de concurrencia introduce una complejidad adicional al sistema y por tanto solo se debe utilizar cuando no exista una solución de tipo secuencial sencilla que cumpla con los requisitos especificados en el documento SRD.

4.6 Notaciones para el diseño.

El objetivo fundamental de cualquier notación es resultar precisa, clara y sencilla de interpretar, evitando ambigüedades.

Según el aspecto del sistema que se trata de describir tenemos varios tipos:

4.6.1 Notaciones estructurales

Estas notaciones sirven para cubrir un primer nivel del diseño arquitectónico. Se trata de desglosar y estructurar el sistema en sus partes fundamentales. Podemos diferenciar dos notaciones habituales para desglosar un sistema en sus partes:

Diagrama de bloques: en la figura 4.1 de la pagina 133 se muestra el diagrama de bloques de un sistema que está dividido en cuatro bloques y en el que además se indican las conexiones que existen entre ellos. En algunos casos estas conexiones también pueden reflejar una cierta clasificación o jerarquía de los bloques.

Cajas adosadas: en la figura 4.2 de la pagina 134 se emplean “cajas adosadas” para delimitar los bloques. La conexión entre los bloques se pone de manifiesto cuando entre dos cajas existe una frontera común.

4.6.1.1 Diagramas de estructura

Estos diagramas fueron propuestos para describir la estructura de los sistemas software como una jerarquía de subprogramas o módulos en general. En la Figura 4.3 de la página 135 se muestra un Diagrama de Estructura.

En este diagrama podemos observar las siguientes figuras:

RECTÁNGULOS: representa un módulo o subprograma cuyo nombre se indica en su interior.

LÍNEA: une a dos rectángulos e indica que el módulo superior llama o utiliza al módulo inferior. A veces la línea acaba en una flecha junto al módulo inferior al que apunta.

ROMBO: se sitúa sobre una línea e indica que esa llamada es opcional. Se puede prescindir de este símbolo si en la posterior descripción del módulo superior, mediante pseudocódigo u otra notación, se indica que el módulo inferior se utiliza sólo opcionalmente.

ARCO: se sitúa sobre una Línea e indica que esa llamada se efectúa de manera repetitiva. Se puede prescindir de ese símbolo si en la posterior descripción del módulo superior, mediante pseudocódigo u otra notación, se indica que el módulo inferior se utiliza de forma repetitiva.

CIRCULO CON FLECHA: se sitúa en paralelo a una Línea y representa el envío de los datos, cuyo nombre acompaña al símbolo, desde un módulo a otro. El sentido del envío lo marca la flecha que acompaña al círculo. Para indicar que los datos son una información de control se utiliza un círculo relleno. Una información de control sirve para indicar SI/NO, o bien un estado: Correcto / Repetir / Error/ Espera / Desconectado /…

El Diagrama de Estructura no establece ninguna secuencia concreta de utilización de los módulos y tan solo refleja una organización estática de los mismos.

4.6.1.2 Diagramas HIPO

Notación propuesta por IBM para facilitar y simplificar el diseño y desarrollo de sistemas software fundamentalmente de gestión (facturación, contabilidad, etc.).

Podemos destacar que:

La mayoría de estos sistemas se pueden diseñar como una estructura jerarquizada de subprogramas o módulos.

El formato de todos los módulos se puede adaptar a un mismo patrón formado por:

- Los datos de entrada (Input).
- El tipo de proceso que se realiza con los datos (Process).
- Los resultados de salida que proporciona (Output).

Dentro de los Diagramas HIPO se diferencian dos diagramas que son:

- El Diagrama HIPO de Contenidos: se utiliza para establecer la jerarquía de los módulos del sistema. Cada módulo tiene un nombre (A, B, C,…) y una referencia al correspondiente Diagrama HIPO de Detalle (0. 0, 1.0 , …). Ver figura 4.4 página 137.
- El Diagrama HIPO de Detalle: constan de 3 zonas :
    - Entrada (I).
    - Proceso (P).
    - Salida (O).

En las zonas de Entrada y Salida se indican respectivamente los datos que entran y salen del módulo.

En la zona central se detalla el pseudocódigo del proceso con referencia a otros diagramas de Detalle de nivel inferior en la jerarquía.

La lista de los diagramas referenciado se listan a continuación de la partícula PARA, por la parte superior.

A continuación de la partícula DE se indica el diagrama de detalle de nivel superior N (i,j).

4.6.1.3 Diagramas de Jackson

Con este Diagrama se diseñada sistemas software a partir de las estructuras de sus datos de entrada y salida.

El proceso se lleva cabo en tres pasos:

Especificación de las estructuras de datos de entrada y salida.

Obtención de una estructura del programa capaz de transformar las estructuras de datos de entrada en las de salida. Este paso implica una conversión de las estructuras de datos en las correspondientes estructuras de programa que las manejan, teniendo en cuenta las siguientes equivalencias:

TUPLA – SECUENCIA: colección de elementos de tipo diferentes combinados en un orden fijo.

UNION – SELECCIÓN: selección de un elemento entre varios posibles, de tipos diferentes.

FORMACION – ITERACION: colección de elementos del mismo tipo.

Expansión de la estructura del programa para lograr el diseño detallado del sistema. Para realizar este paso se suele utilizar Pseudocódigo.

La metodología Jackson está englobada dentro de las de Diseño Dirigido por los Datos, que ha sido utilizado fundamentalmente para diseñar sistemas relativamente pequeños de procesamiento de datos.

4.6.2 Notaciones estáticas

Sirven para describir características estáticas del sistema tales como la organización de la información, sin tener en cuenta su evolución durante el funcionamiento del sistema.

En el documento SRD de especificación se realiza una propuesta de organización a grandes rasgos y de acuerdo a las necesidades externas del sistema. En la fase de diseño se completa la organización de la información con los aspectos internos: datos auxiliares, mayor eficiencia en el manejo de datos, datos redundantes etc.

Resumiendo: como resultado del diseño se tendrá una organización de la información con un nivel de detalle mucho mayor. Las notaciones empleadas son las mismas que las empleadas en la especificación.

Diccionario de datos: con esta notación se detalla la estructura interna de los datos que maneja el sistema. Para el diseño se parte del diccionario de datos incluido en el SRD y mediante refinamientos se alcanza el nivel de detalle para empezar la codificación.

Diagramas Entidad - Relación: esta notación permite definir el modelo de datos, las relaciones entre los datos y en general la organización de la información. Para la fase de diseño se parte del diagrama propuesto en el SRD y se amplía con las nuevas entidades y sus relaciones que aparecen en la fase de diseño.

4.6.3 Notaciones dinámicas

Permiten describir el comportamiento del sistema durante su funcionamiento. La especificación de un sistema es una descripción del comportamiento requerido desde un punto de vista externo. Al diseñar la dinámica del sistema se detallara su comportamiento externo y se añadirá la descripción de un comportamiento interno capaz de garantizar que se cumplen todos los requisitos especificados en el documento SRD. Las notaciones que se emplean para el diseño son las mismas utilizadas en la especificación.

Diagrama de flujo de datos: desde el punto de vista del Diseño serán más exhaustivos que los de la especificación (SRD).

Diagrama de transición de estados: en el diseño del sistema pueden aparecer nuevos diagramas de estado que reflejen las transiciones entre estados internos. Es conveniente no ampliar o modificar los recogidos en el SRD.

Lenguaje de Descripción de Programas (PDL): Sirve tanto para la especificación funcional del sistema como para elaborar el diseño del mismo. Sin embargo, si se quiere descender al nivel de detalle que se requiere en la fase de diseño, la notación PDL se amplía con ciertas estructuras de algún lenguaje de alto nivel.

4.6.4 Notaciones híbridas

Estas notaciones tratan de cubrir simultáneamente aspectos estructurales, estáticos y dinámicos. Entre ellas podemos destacar las siguientes:

- La metodología de Jackson.
- Diseño orientado a los datos de Warnier.
- Diseño basado en abstracciones.
- Diseño orientado a objetos.

4.6.4.1 Diagramas de abstracciones

En una abstracción se distinguen tres partes:

- ***Nombre***: es el identificador de la abstracción
- ***Contenido***: es el elemento estático de la abstracción y en él se define la organización de los datos que constituyen la abstracción. En Modula son las definiciones de tipos, constantes y variables declaradas en el módulo de definición.
- ***Operaciones***: es el elemento dinámico de la abstracción y en él se agrupan todas las operaciones definidas para manejar el CONTENIDO de la abstracción. En Modula este elemento estaría formado por las definiciones de funciones y/o procedimientos declarados en el módulo de definición.

Podemos distinguir tres tipos de abstracciones:

Abstracción Funcional: un subprograma constituye una operación abstracta que denominaremos Abstracción Funcional. Esta forma de abstracción no tiene la parte de Contenido y solo está constituida por una única Operación.

Los tipos Abstractos de Datos: al Analizar y Diseñar un sistema se identifican datos de diferentes tipos con los que hay que operar. Se puede agrupar en una misma entidad la estructura del tipo de datos con las correspondientes operaciones necesarias para su manejo. A esta forma de abstracción la denominamos Tipo Abstracto de Datos y tiene una parte CONTENIDO y también sus correspondientes operaciones.

Los datos encapsulados: cuando solo se necesita una variable de un determinado tipo abstracto, su declaración se puede encapsular dentro de la misma abstracción. De esta forma, todas las operaciones de la abstracción se referirán siempre a esa variable sin necesidad de indicarlo de manera explícita. A esta forma de abstracción la denominamos Dato Encapsulado y tiene CONTENIDO y OPERACIONES, pero no permite declarar otras variables de su mismo tipo.

Ver Figura 4.9. Página 145.

4.6.4.2 Diagramas de objetos

La metodología de diseño basado en abstracciones y la metodología orientada a los objetos se han desarrollado de forma paralela pero en ámbitos muy distintos. Aunque las similitudes entre ambos conceptos son muy grandes, su desarrollo en distintos ámbitos ha propiciado la utilización de una terminología distinta para indicar lo mismo.

La estructura de un objeto es exactamente igual que la estructura de una abstracción. En cuanto a la terminología empleada se pueden establecer las siguientes equivalencias:

ABSTRACCIONES	OBJETOS
Tipo Abstracto de Datos	Clase de Objeto
Abstracción Funcional	NO HAY EQUIVALENCIA
Dato Encapsulado	NO HAY EQUIVALENCIA
Dato Abstracto (Variable o Constante)	Objeto (Ejemplar de Clase)
Contenido	Atributos
Operaciones	Métodos
Llamada a una Operación	Mensaje al Objeto
Equivalencia de Terminologías	

Además solo entre objetos se contempla una relación de herencia.

Si consideramos un objeto formado solo por sus atributos tendremos una estructura de datos.

Todas las entidades en un modelo de datos son objetos (o más exactamente, clases de objetos).

Podemos establecer dos tipos de relaciones especiales entre objetos:

Clasificación, especialización o herencia (no valida en abstracciones): en este caso los objetos hijos adaptan las operaciones heredadas a sus necesidades concretas. En la Figura 4.11 de la Página 148 se muestra un ejemplo de herencia entre objetos.

La notación empleada es la sugerida por la metodología OMT (Object Modelling Technique). Con el **Triangulo** se indica que los objetos inferiores (Hijo_Uno, Hijo_Dos e Hijo_Tres) heredan los atributos y las operaciones del objeto superior (Padre).

Con la relación de herencia no es necesario indicar la cardinalidad entre las clases de objetos, que está implícita en el diagrama, ya que en él aparece expresamente cada relación de herencia entre clases.

Composición (valida en abstracciones): en este caso se permite describir un objeto mediante los elementos que lo forman. En la Figura 4.12 de la Página 149 se muestra la relación de composición

con la notación OMT. El **Rombo** indica que el objeto Estructura está compuesto de Elemento_1, Elemento_2 y Elemento_3.

En la relación de Composición solo hay que indicar la cardinalidad en un sentido.

Cada objeto hijo solo puede formar parte de un objeto padre. Solo falta indicar qué número mínimo y máximo de objetos de cada clase hija se pueden utilizar para componer un objeto de la clase padre. Como se muestra en la Figura 4.12 la cardinalidad se indica junto a cada uno de los objetos componentes. Así, para formar Estructura son necesarios cero o varios Elemento_1, uno y solo un Elemento_2 y al menos un Elemento_3.

4.7 Documentos de diseño.

El resultado de la etapa de diseño se recoge en un documento que se utilizara como elemento de partida para las sucesivas etapas del proyecto y que denominaremos **Documento de Diseño de Software (SDD).**

Cuando la complejidad del sistema haga que el documento SDD resulte muy voluminoso y difícil de manejar, es habitual utilizar dos a más documentos para describir de forma jerarquizada la estructura global del sistema.

Las normas de la **ESA** establecen el empleo de un **Documento de Diseño Arquitectónico (ADD)** para describir el sistema en su conjunto y otro **Documento de Diseño Detallado (DDD)** para describir por separado cada uno de los componentes del sistema.

4.7.1 Documento de Diseño Arquitectónico (ADD)

Consta de las siguientes partes (Ver Cuadro 4.2 Pág. 151):

4.7.1.1 Introducción:

Se da una visión general de todo el documento.

Consta de los mismos apartados que el SRD (Objetivo, Ámbito, Definiciones, Siglas y Abreviaturas, y Referencias), pero referidos al sistema tal como se ha diseñado.

Siempre que se considere interesante se debe hacer referencia al SRD.

4.7.1.2 Panorámica Del Sistema:

Se da una visión general de los requisitos funcionales (y de otro tipo) del sistema que ha de ser diseñado, haciendo referencia al documento SRD.

4.7.1.3 Contexto del Sistema:

Definición de interfaz externa:

Se indica si el sistema posee conexiones con otros sistemas y si debe funcionar de una forma integrada con ellos.

En cada apartado se define la interfase que se debe utilizar con cada uno de os otros sistemas.

Si el sistema no necesita intercambiar información con ningún otro se indicará "No existe interfaz" o “No aplicable".

4.7.1.4 Diseño del Sistema:

Se considera el sistema en su conjunto y se hace una primera estructuración en componentes.

Metodología de diseño de alto nivel:

> Se describe brevemente la metodología a seguir en el proceso de diseño de la arquitectura del sistema.

Descomposición del sistema:

> Se describe el primer nivel de descomposición del sistema en sus componentes principales.
>
> Se enumeran los componentes y las relaciones estructurales entre ellos.

4.7.1.5 Diseño de los Componentes

Las siguientes subsecciones se repiten para cada uno de los componentes mencionados en el apartado **Descomposición del sistema.**

Identificador del Componente:

Nombre del Componente.

Dos Componentes no pueden tener nunca el mismo nombre.

El nombre se elegirá tratando que refleje su naturaleza.

Tipo

Se describe la clase de componente. En algunos casos basta con indicar el tipo de componente: subprograma, modulo, procedimiento, proceso, datos, etc.

Es posible definir nuevos tipos basados en otros más elementales.

Dentro del mismo documento se debe establecer una lista coherente de los tipos usados.

Objetivo

Se debe describir la necesidad de que exista el componente haciendo referencia a un requisito concreto que se trata de cubrir por ejemplo.

Si el requisito no forma parte del documento SRD se tendrá que detallar en este momento.

Función

Se describe qué hace el componente.

Esto se puede detallar mediante la transformación entrada/salida que realiza o si el componente es un dato se describirá qué información guarda.

Subordinados

Se enumeran todos los componentes usados por éste.

Dependencias

Se enumeran los componentes que usan a éste.

Junto a cada dependencia se podrá indicar su naturaleza: invocación de operación, datos compartidos, inicialización , creación , etc.

Interfaces

Se describe cómo otros componentes interactúan con éste.

Se tienen que establecer las distintas formas de interacción y las reglas para cada una de ellas: paso de parámetros, zona común de memoria, mensajes, etc.

También se indicaran las restricciones tales como rangos, errores, etc.

Recursos:

Se describen los elementos usados por este componente que son externos a este diseño: impresoras, particiones de disco, organización de la memoria, etc.

Referencias

Se presentan todas las referencias utilizadas.

Proceso

Se describen los algoritmos o reglas que utiliza el componente para realizar su función, como un refinamiento de la sección Función.

Datos

Se describen los datos internos del componente incluyendo el método de representación, valores iniciales, formato, valores validos, etc.

Esta descripción se puede realizar mediante un diccionario de datos.

Además se indicará el significado de cada elemento.

4.7.1.6 Viabilidad y Recursos Estimados.

Se analiza la viabilidad del sistema y se concretan los recursos que se necesitan para llevarlo a cabo.

4.7.1.7 Matriz Requisitos/Componentes.

Se muestra una matriz poniendo en las filas todos los requisitos y en las columnas todos los componentes.

Para cada requisito se marcará el componente o componentes encargados de que se cumpla.

4.7.2 Documento de Diseño Detallado

Este documento irá creciendo con el desarrollo del proyecto. En proyectos grandes puede ser conveniente organizarlo en varios volúmenes.

Como se puede ver en la página 155, cuadro 4.3, el formato de este documento es bastante similar al ADD. La diferencia entre ambos es el nivel de detalle al que se desciende.

En la Parte 1, Sección del documento se recogen todas las normas, convenios y procedimientos de trabajo que se deben aplicar durante el desarrollo del sistema.

De esta Sección depende que el trabajo realizado por un equipo amplio de personas tenga una estructura coherente y homogénea.

La realización de esta sección debe ser la primera actividad del diseño detallado antes de iniciar el diseño propiamente dicho.

Si se utilizan las mismas normas en diversos proyectos, se puede sustituir esta sección por referencias a los documentos que contienen la información correspondiente.