

TEMA 3 FUNDAMENTOS DEL DISEÑO DE SOFTWARE

Las etapas de desarrollo se dedican a determinar CÓMO se debe resolver el proyecto.

3.1 INTRODUCCIÓN

¿Qué se entiende por diseño de software?: Sería la descripción del sistema a desarrollar. Se trata de definir y formalizar la estructura del sistema con el suficiente detalle como para permitir su realización física.

El punto de partida es el documento de especificación de requisitos (SRD).

Es muy importante la experiencia previa, siempre que sea posible se tratará de reutilizar el mayor número posible de módulos o elementos ya desarrollados. Cuando no sea posible, por tratarse del diseño de un sistema completamente original, seguro que para comenzar el diseño, al menos se podrá aprovechar el enfoque dado a algún otro proyecto anterior. Se conoce como aprovechar el know-how (saber hacer). En sucesivas iteraciones se perfilará el enfoque más adecuado para el nuevo diseño.

La etapa de diseño es la más importante de la fase de desarrollo de software. En esta etapa se tiene que pasar de una forma gradual de QUÉ debe hacer el sistema, según se detalla en el documento de requisitos, al CÓMO lo debe hacer.

Durante el diseño se tiene que pasar 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:

1. DISEÑO ARQUITECTÓNICO.

Abordar aspectos estructurales y de organización del sistema y su posible división en subsistemas o módulos. Establecer las relaciones entre los subsistemas creados y definir las interfaces entre ellos.

2. DISEÑO DETALLADO.

Aborda la organización de los módulos. Trata de determinar cuál es la estructura más adecuada para cada uno de los módulos.

Como resultado del diseño detallado, aparecen nuevos módulos que se deben incorporar al diseño global, componentes que es razonable agrupar en un único módulo, o módulos que desaparecen por estar vacíos de contenido. Existe una interrelación entre el diseño arquitectónico y el detallado.

3. DISEÑO PROCEDIMENTAL

Abordar la organización de operaciones o servicios que ofrecerá cada uno de los módulos. En esta actividad se detallan en pseudocódigo o PDL solamente los aspectos más relevantes de cada algoritmo.

4. DISEÑO DE DATOS

Aborda la organización de la base de datos del sistema, se puede realizar en paralelo con el diseño detallado y procedimental. El punto de partida para esta actividad es el diccionario de datos y los diagramas E-R de la especificación del sistema.

Es de gran importancia conseguir que el objetivo de que el sistema sea reutilizable y fácilmente mantenible.

5. DISEÑO DE LA INTERFAZ DE USUARIO

Aborda la organización de la interfaz de usuario. La importancia de esta actividad ha propiciado el desarrollo de técnicas y herramientas específicas que facilitan mucho el diseño.

El resultado de todas estas actividades de diseño debe dar lugar a una especificación lo más formal posible de la estructura global del sistema y de cada uno de los elementos del mismo, y se recogerá en el Documento de Diseño de Software (SDD: Software Design Document o Software Design Description).

3.2 CONCEPTOS BASE

ABSTRACCIÓN

Cuando se diseña un nuevo software es importante identificar los elementos realmente significativos de los que consta y abstraer la utilidad específica de cada uno más allá del sistema software para el que se está diseñando.

Dos de los principales objetivos del diseño son:

- Conseguir elementos fácilmente reutilizables.
- Conseguir elementos fácilmente mantenibles.

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

- Abstracciones funcionales

Sirven para crear expresiones parametrizadas o acciones mediante el empleo de funciones o procedimientos.

Es necesario fijar los parámetros o argumentos que se le deben pasar, el resultado que devolverá, lo que pretende que resuelva y como lo debe resolver.

- Tipos abstractos

Sirven para crear los nuevos tipos de datos que se necesitan para abordar el diseño del sistema. Junto al nuevo tipo de dato se deben diseñar todos los métodos u operaciones que se pueden realizar con él.

- Máquinas abstractas

Permite establecer un nivel de abstracción superior a los anteriores. Se define de una manera formal el comportamiento de una máquina.

MODULARIDAD

Ventajas de utilizar un diseño modular:

- División del trabajo

Permite encargar a personas diferentes el desarrollo de cada módulo y que todas ellas puedan trabajar simultáneamente.

- Claridad

Más fácil entender y manejar cada una de las partes o módulos de un sistema que tratar de entenderlo como un todo compacto.

- Reducción de costos.

Es más barato desarrollar, depurar, documentar, probar y mantener un sistema modular que otro que no lo es.

Tener en cuenta que si el número de módulos crece innecesariamente esta afirmación puede no ser correcta.

- Reutilización.

Los módulos se diseñan teniendo en cuenta otras posibles aplicaciones.

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.

REFINAMIENTO

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 sistema software expresado en su especificación se debe refinar en sucesivos pasos hasta que todo quede expresado en el lenguaje de programación del computador.

El uso directo e irrestringido de refinamientos sucesivos conduciría a programas monolíticos. El refinamiento directo debe dejar de aplicarse si el tamaño del programa resultante para una sola acción global excede de un límite razonable.

Si se excede este límite, convendrá aplicar las ideas de abstracción u modularidad para fragmentar una operación global en varias separadas.

ESTRUCTURA DE DATOS

Estructuras fundamentales:

- Registros
- Conjuntos
- Formaciones
- Listas, pilas, colas
- Árboles
- Grafos
- Tablas
- Ficheros

A partir de estas estructuras básicas se debe buscar la combinación más adecuada para lograr aquella estructura o estructuras que den respuesta a las necesidades del sistema.

OCULTACIÓN

Trata de ocultar al “usuario” todo lo pueda ser susceptible de cambio en el futuro y además es irrelevante para el uso.

Ventajas de aplicar este concepto:

- **DEPURACIÓN**

Es más sencillo detectar qué módulo concreto no funciona correctamente.

- **MANTENIMIENTO**

Cualquier operación de mantenimiento en un módulo concreto no afectará al resto de los módulos del sistema.

GENERICIDAD

Agrupar aquellos elementos del sistema que utilizan estructuras semejantes o que necesitan de un tratamiento similar.

Diseñar un elemento genérico con las características comunes a todos los elementos agrupados. Posteriormente, cada uno de los elementos agrupados se pueden diseñar como un caso particular del elemento genérico.

HERENCIA

Establecer una clasificación o jerarquía entre 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.

Los elementos “hijos” pueden tener otros “hijos” que hereden de ellos de una forma semejante.

La herencia permite reutilizar una gran cantidad de software ya desarrollado.

POLIMORFISMO

Conseguir que un mismo elemento software adquiriera varias formas simultáneamente.

Distintas posibilidades para conseguir polimorfismo:

- El concepto de **genericidad** es una manera de lograr que un elemento genérico pueda adquirir distintas formas cuando se particulariza su utilización.
- Polimorfismo de **anulación**. Las estructuras y operaciones heredadas se pueden adaptar a las necesidades concretas del "hijo".
- Polimorfismo **diferido**. Plantea la necesidad de la operación para el elemento "padre", pero su concreción se deja diferida para que cada uno de los elementos "hijos" concrete su forma específica.
- Polimorfismo de **sobrecarga**. Quienes adquieren múltiples formas son los operadores, funciones o procedimientos.

El concepto de polimorfismo está ligado a las metodologías orientadas a objetos.

CONCURRENCIA

En los sistemas de tiempo real existe la necesidad de aprovechar toda la capacidad de proceso del computador para atender a todos los eventos que se producen y en mismo instante en que se producen. Se establecen tiempos máximos de respuesta ante determinados eventos (alarmas, fallos, errores, etc.)

Cuando se trata de diseñar un sistema con restricciones de tiempo se debe tener en cuenta lo siguiente:

1. TAREAS CONCURRENTES

Determinar qué tareas se deben ejecutar en paralelo para cumplir con las restricciones impuestas. Prestar especial atención a tareas con tiempos de respuesta más críticos y aquellas otras que se ejecutarán con mayor frecuencia.

2. SINCRONIZACIÓN DE TAREAS

Determinar los puntos de sincronización entre las distintas tareas. Para la sincronización se utilizan semáforos o mecanismos de tipo rendezvous disponibles en los sistemas operativos o en lenguajes de programación concurrentes.

3. COMUNICACIÓN ENTRE TAREAS

La cooperación se basa en el empleo de datos compartidos o mediante el paso de mensajes entre tareas. Se debe concretar qué tareas serán las productoras y qué otras tareas serán las consumidoras.

En el caso de usar datos compartidos se debe evitar que los datos puedan ser modificados en el momento de la consulta. Se usan mecanismos como semáforos,

monitores, regiones críticas, etc. Para garantizar la exclusión mutua entre las distintas tareas que modifican y consultan los datos compartidos

4. INTERBLOQUEOS (deadlock)

Estudiar los posibles interbloqueos entre tareas. Un interbloqueo se produce cuando una o varias tareas permanecen esperando por tiempo indefinido una situación que se puede producir nunca.

El concepto de concurrencia introduce una complejidad adicional al sistema. Sólo se debe utilizar cuando no exista una solución de tipo secuencial sencilla que cumpla con los requisitos especificados.

3.3 NOTACIONES PARA EL DISEÑO

El objetivo fundamental de cualquier notación es resultar precisa, clara y sencilla de interpretar en el aspecto concreto que describe. Evitar ambigüedades e interpretaciones erróneas del diseño.

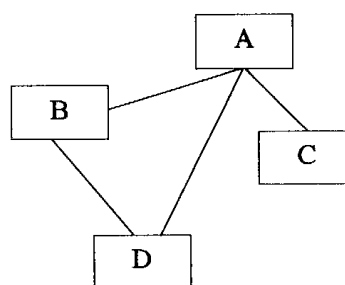
Según el aspecto del sistema que se trata de describir es aconsejable emplear una u otra clase de notación. Se pueden clasificar las notaciones en los siguientes grupos:

- Notaciones estructurales
- Notaciones estáticas o de organización
- Notaciones dinámicas o de comportamiento
- Notaciones híbridas

NOTACIONES ESTRUCTURALES

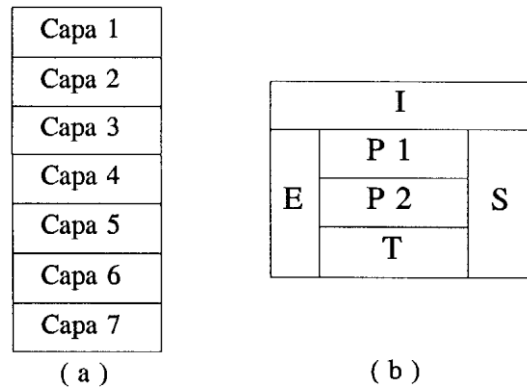
Sirven para cubrir un primer nivel del diseño arquitectónico. Se trata de desglosar y estructurar el sistema en sus partes fundamentales.

Una notación habitual para desglosar el sistema es el empleo de diagramas de bloques.



Se indican las conexiones que existen entre los diferentes bloques. En algunos casos estas conexiones también pueden reflejar una cierta clasificación o jerarquía de los bloques.

Otra notación para estructurar un sistema es emplear “cajas adosadas”. La conexión entre los bloques se pone de manifiesto cuando entre dos cajas existe una frontera común.



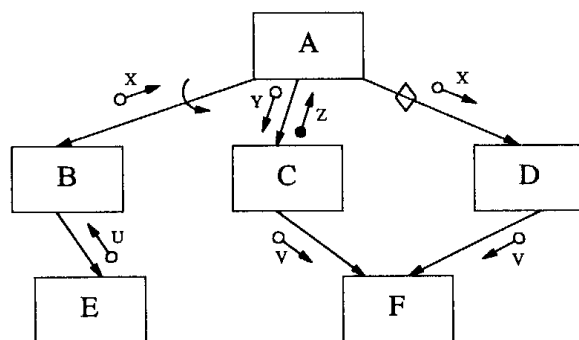
El diagrama (a) muestra un sistema organizado por capas. Cada capa es un subsistema. Entre capas adyacentes está definida una interfaz completamente estándar. Posibilita que las modificaciones en la implementación de una capa nunca afecten al resto.

En el diagrama (b), los subsistemas generales propuestos en la plantilla son:

- E (entrada)
- S (salida)
- T (test para la autocomprobación y el mantenimiento)
- I (interfase de usuario)
- P1,P2,... (procesado y control del sistema)

DIAGRAMAS DE ESTRUCTURA

Propuestos por Yourdon y Myers, para describir la estructura de los sistemas software como una jerarquía de subprogramas o módulos en general.



El significado de los símbolos es el siguiente:

- **Rectángulo.** 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.

- **Rombo.** Se sitúa sobre una línea e indica que esa llamada o utilización es opcional.
- **Arco.** Se sitúa sobre una línea e indica que esa llamada o utilización se efectúa de manera repetitiva.
- **Círculo 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 de envío lo marca la flecha. 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 / ...

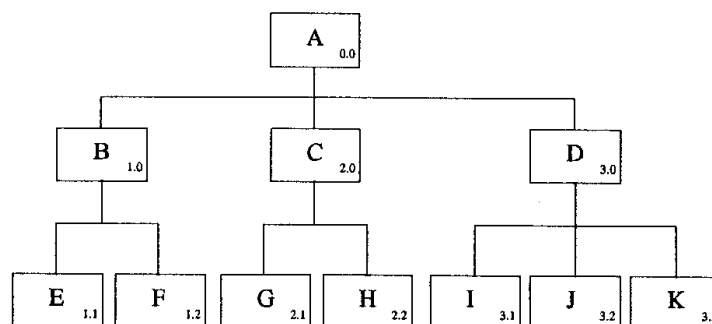
El resultado es un diagrama en forma de árbol mostrando la jerarquización de los módulos. Es normal que varios módulos superiores utilicen un mismo módulo inferior.

El diagrama de estructura no establece ninguna secuencia concreta de utilización de los módulos. Refleja una organización estática de los mismos.

DIAGRAMAS HIPO

HIPO (Hierarchy-Input-Process-Output).

Notación para facilitar y simplificar el diseño y desarrollo de sistemas software, destinados fundamentalmente a gestión. El formato de todos los módulos se puede adaptar a un mismo patrón caracterizado por los datos de entrada (input), el tipo de proceso (process), que se realiza con los datos y los resultados de salida que proporciona (output).



La figura muestra un diagrama HIPO de contenidos. Se utiliza para establecer la jerarquía entre los módulos del sistema. Es asimilable a un diagrama de estructura simplificado en el que no se indican los datos que se intercambian los módulos. Cada módulo tiene un nombre (A, B, C, ...) y una referencia al correspondiente diagrama HIPO de detalle (0.0, 1.0, ...).

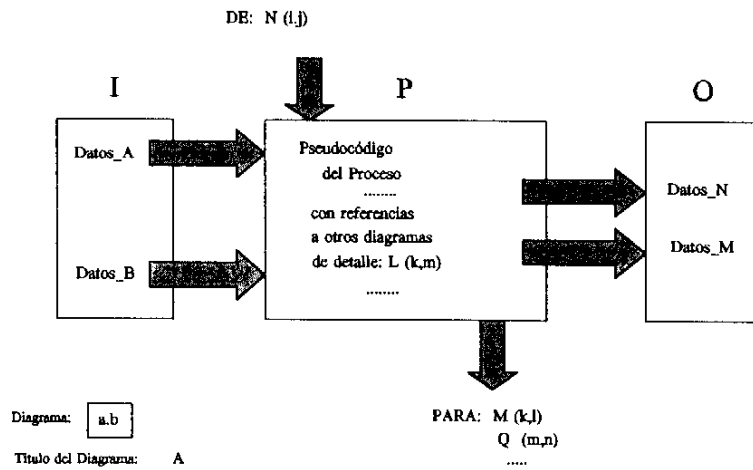


Diagrama HIPO de detalle. Los diagramas de detalle constan de tres zonas: Entradas (I), Proceso (P), Salida (O).

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 referenciados se listan a continuación de la partícula PARA: en la parte superior y a continuación de la partícula DE: se indica el diagrama de detalle de nivel superior: N(i,j).

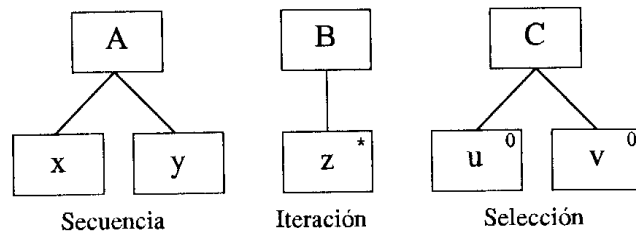
DIAGRAMAS DE JACKSON

Diseñar sistemas software a partir de las estructura de sus datos de entrada y salida. El proceso de diseño es bastante sistemático y se lleva acabo en tres pasos:

1. Especificación de las estructuras de los datos de entrada y salida.
2. Obtención de un estructura del programa capaz de transformar las estructuras de datos de entrada en las de salida. Implica una conversión de las estructuras de datos en las correspondientes estructuras de programa que las manejan.
Equivalencias clásicas:
 - a. TUPLA – SECUENCIA: colección de elementos de tipos diferentes, combinados en un orden fijo.
 - b. UNIÓN – SELECCIÓN: selección de un elemento entre varios posibles, de tipos diferentes.
 - c. FORMACIÓN – ITERACIÓN: colección de elementos del mismo tipo.
3. Expansión de la estructura del programa para lograr el diseño detallado del sistema. Normalmente se utiliza pseudocódigo.

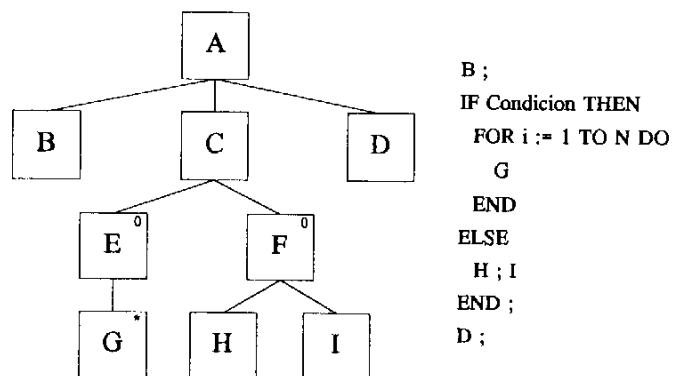
La metodología Jackson está englobada dentro de las de Diseño Dirigido por los Datos.

Notación propuesta:



La estructura del elemento superior se detalla según queda indicado por los elementos inmediatamente inferiores. Es fundamental la situación de cada elemento en el diagrama.

En la figura el elemento A es igual a la secuencia x-y leída de izquierda a derecha y nunca a la inversa. El elemento B es igual a la repetición de cero o más veces del elemento z (indicado por el símbolo “*”). El elemento C es igual a una selección entre los elementos u y v (indicado por el símbolo “0”).



Si los elementos del diagrama representan datos, la estructura equivalente de diccionario de datos es:

$$A = B + C + D$$

$$C = (E \mid F)$$

$$E = \{ G \}$$

$$F = H + I$$

NOTACIONES ESTÁTICAS

Sirven para describir características estáticas del sistema, sin tener en cuenta su evolución durante el funcionamiento del sistema.

Como resultado del diseño se tendrá una organización de la información con un nivel de detalle mucho mayor. Las notaciones que se pueden emplear para describir el resultado de este trabajo son las mismas que se emplean para realizar la especificación.

DICCIONARIO DE DATOS

Detalla la estructura interna de los datos que maneja el sistema. Para el diseño se partirá del diccionario de datos incluido en el documento SRD y mediante los refinamientos necesarios se ampliará y completará hasta alcanzar el nivel de detalle necesario para iniciar la codificación.

DIAGRAMAS ENTIDAD-RELACIÓN

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 tomará como punto de partida el diagrama propuesto en el documento SRD, que se completará y ampliará con las nuevas entidades y relaciones entre las mismas, que aparezcan en la fase de diseño del sistema.

NOTACIONES DINÁMICAS

Permiten describir el comportamiento del sistema durante su funcionamiento. Al diseñar la dinámica del sistema se detallará 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.

DIAGRAMAS DE FLUJO DE DATOS

Desde un punto de vista de diseño, los diagramas de flujo de datos serán mucho más exhaustivos que los de la especificación, debido a la necesidad de describir cómo se hacen internamente las cosas y no sólo qué cosas debe hacer el sistema.

DIAGRAMAS DE TRANSICIÓN DE ESTADOS

En el diseño del sistema pueden aparecer nuevos diagramas de estado que reflejen las transiciones entre estados internos. Es preferible no modificar o ampliar los diagramas recogidos en el documento SRD encargados de reflejar el funcionamiento externo del sistema.

LENGUAJE DE DESCRIPCIÓN DE PROGRAMAS (PDL)

Esta notación se utiliza tanto para realizar la especificación funcional del sistema como para elaborar el diseño del mismo. La diferencia entre ambas situaciones la marca el nivel de detalle al que se desciende en la descripción funcional.

Tanto al especificar como al diseñar se utilizan las mismas estructuras básicas de la notación PDL. Cuando se quiere descender el 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. (Ada-PDL).

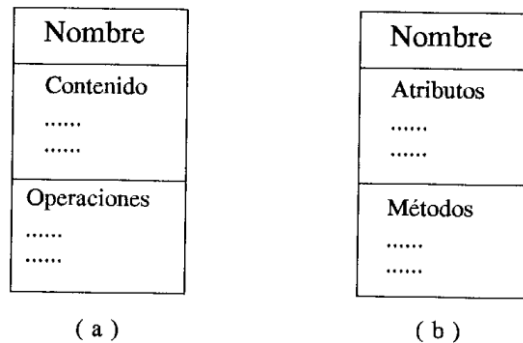
NOTACIONES HÍBRIDAS

Tratan de cubrir simultáneamente aspectos estructurales, estáticos y dinámicos.
Metodologías de diseño basado en abstracciones y diseño orientado a objetos.

DIAGRAMA DE ABSTRACCIONES

Describir la estructura de un sistema software compuesto por elementos abstractos.

Diagrama de una estructura (a) y de un objeto (b)



En una abstracción se distinguen tres partes o elementos:

- NOMBRE: identificador de la abstracción.
- CONTENIDO: elemento estático de la abstracción y en él se define la organización de los datos que constituyen la abstracción.
- OPERACIONES: elemento dinámico de la abstracción y en él se agrupan todas las operaciones definidas para manejar el contenido de la abstracción.

Inicialmente la única forma de abstracción disponible en los lenguajes, y por tanto con la que únicamente se podía abordar un diseño, era la definición de subprogramas: funciones (expresiones parametrizadas) o procedimientos (acciones parametrizadas). Un subprograma constituye una operación abstracta que denominaremos abstracción funcional. Esta forma de abstracción no tiene la parte de contenido y sólo está constituida por una única operación.

En un diseño bien organizado se puede agrupar en una misma entidad la estructura del tipo de datos con las correspondientes operaciones necesarias para su manejo. Esta forma de abstracción se denomina **tipo abstracto de datos** y tiene una parte de contenido y también sus correspondientes operaciones.

Cuando sólo se necesita una variable de un determinado tipo abstracto su declaración se puede encapsular dentro de la misma abstracción. Así, todas las operaciones de la abstracción se referirán siempre a esa variable sin necesidad de indicarlo de manera explícita. Esta forma de abstracción se denomina **Dato encapsulado**, tiene contenido y operaciones, al igual que un tipo abstracto, pero no permite declarar otras variables de su mismo tipo.

Notación gráfica para indicar la forma de abstracción que se está empleando:

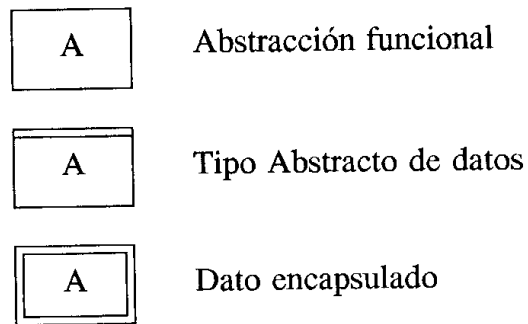
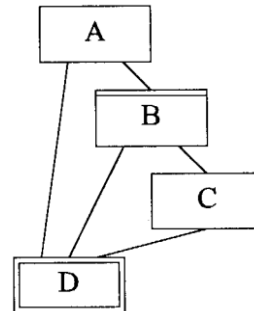


Diagrama de estructura de un sistema:



Los módulos son abstracciones en sus distintas formas posibles y la relación entre las abstracciones es jerárquica e implica que la abstracción superior utiliza a la inferior. La abstracción funcional A utiliza el dato encapsulado D y el tipo abstracto B. Al dato encapsulado D también lo utilizan B y la abstracción funcional C. A su vez, la abstracción funcional C también es utilizada por B.

DIAGRAMAS DE OBJETOS

Las abstracciones se pueden considerar una propuesta de los expertos en programación, mientras que los objetos son una propuesta de los expertos en inteligencia artificial.

La estructura de un objeto es exactamente igual que la estructura de una abstracción. Las diferencias fundamentales entre ambos conceptos son las siguientes:

1. No existe nada equivalente a los datos encapsulados ni a las abstracciones funcionales cuando se utilizan objetos en forma estricta.
2. Sólo entre objetos se contempla una relación de herencia.

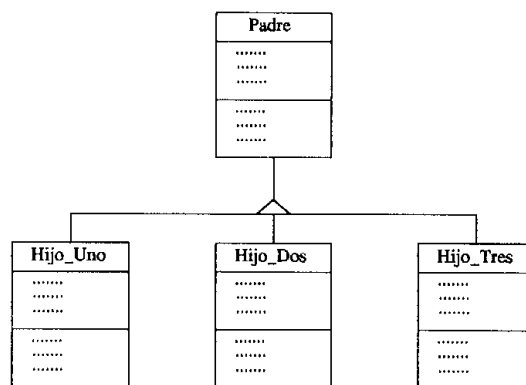
Equivalencias de terminologías:

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 la clase)
Contenido	Atributos
Operaciones	Métodos
Llamada a una operación	Mensaje al objeto

Debido a las propiedades particulares de los objetos, se pueden establecer entre ellos dos tipos de relaciones espaciales:

A. CLASIFICACIÓN, ESPECIALIZACIÓN O HERENCIA

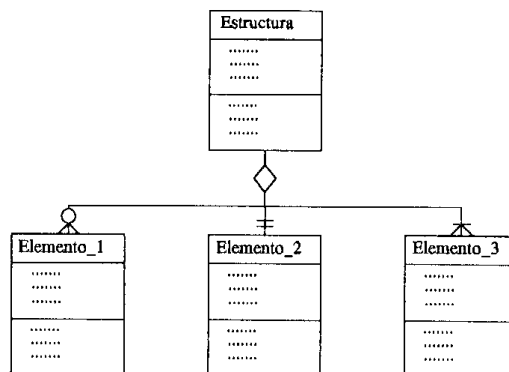
Esta relación entre objetos permite diseñar un sistema aplicando el concepto de herencia. No contemplada entre abstracciones. La herencia también se puede denominar especialización o clasificación.



Con el triángulo se indica que los objetos inferiores heredan los atributos y las operaciones del objeto superior.

B. COMPOSICIÓN

La relación de composición permite describir un objeto mediante los elementos que lo forman. También válida entre abstracciones. El rombo indica que el objeto superior está compuesto por los elementos inferiores.



En la relación de composición sólo hay que indicar la cardinalidad en un sentido. Cada objeto hijo sólo puede formar parte de un objeto padre. El número mínimo y máximo de objetos de cada clase hija se pueden utilizar para componer un objeto de la clase padre. La cardinalidad se indica junto a cada uno de los objetos componentes.

3.4 DOCUMENTOS DE DISEÑO

El resultado principal de la labor realizada en la etapa de diseño se recoge en un documento que se utilizará como elemento de partida para las sucesivas etapas del proyecto, es el **documento de diseño de software** o Software Design Document (SDD)

Cuando la complejidad del sistema haga que el documento SDD resulte muy voluminoso y difícil de manejar es habitual utilizar dos o 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** o Architectural Design Document ADD. Describe el sistema en su conjunto.
- Un **Documento de Diseño Detallado** o Detailed Design Document DDD. Describe por separado cada uno de los componentes del sistema.

DOCUMENTO ADD

Índice del documento ADD :

1. INTRODUCCIÓN

1.1 Objetivo

1.2 Ámbito

1.3 Definiciones, siglas y abreviaturas

1.4 Referencias

2. PANORÁMICA DEL SISTEMA

3. CONTEXTO DEL SISTEMA

3.n Definición de interfaz externa

4. DISEÑO DEL SISTEMA

4.1 Metodología de diseño de alto nivel

4.2 Descomposición del sistema r

5. DISEÑO DE LOS COMPONENTES

5.n Identificador del componente ,

5.n.1 Tipo

5.n.2 Objetivo

5.n.3 Función

5.n.4 Subordinados

5.n.5 Dependencias

5.n.6 Interfase

5.n.7 Recursos

5.n.8 Referencias

5.n.9 Proceso

5.n.10 Datos

6. VIABILIDAD Y RECURSOS ESTIMADOS

7. MATRIZ REQUISITOS/COMPONENTES

Explicación de cada parte del documento ADD:

1. INTRODUCCIÓN

Visión general de todo el documento ADD. Apartados:

- Objetivo.
- Ámbito.
- Definiciones.
- Siglas y abreviaturas.
- Referencias.

Se puede y se debe hacer referencia a los correspondientes apartados del documento SRD.

2. PANORÁMICA DEL SISTEMA

Visión general de los requisitos funcionales y de otro tipo del sistema que ha de ser diseñado, haciendo referencia al documento SRD.

3. CONEXTO DEL SISTEMA

Se indicará si este sistema posee conexiones con otros y si debe funcionar de una forma integrada con ellos. En cada uno de sus apartados se definirá la correspondiente interfase se debe utilizar con cada uno de los otros sistemas. Si el sistema no necesita intercambiar información con ningún otro, se indicará “no existe interfaz” o “No aplicable”.

4. DISEÑO DEL SISTEMA

Describe el nivel superior del diseño, se considera el sistema en su conjunto y se hace una primera estructuración en componentes.

4.1 Metodología de diseño de alto nivel.

Describe brevemente o se hace referencia a la metodología a seguir en el proceso de diseño de la arquitectura.

4.2 Descomposición del sistema.

Primer nivel de descomposición del sistema en sus componentes principales. Se enumeran los componentes y las relaciones estructurales entre ellos.

5. DISEÑO DE LOS COMPONENTES

Los siguientes subprocesos se repiten para cada uno de los componentes mencionados en el apartado 4.2

5.n Identificador del componente

Nombre del componente. Dos componentes no podrán tener nunca el mismo nombre.

5.n.1 Tipo

Describe la clase de componente:

- Subprograma
- Módulo
- Procedimiento
- Proceso de datos
- Etc.

Es posible definir aquí nuevos tipos basados en otros más elementales. Dentro del mismo documento se debe establecer una lista coherente de los tipos usados.

5.n.2 Objetivo

Debe describir la necesidad de que exista el componente. Se puede hacer referencia a un requisito concreto que se trata de cubrir.

5.n.3 Función

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

5.n.4 Subordinados

Enumeran todos los componentes usados por éste.

5.n.5 Dependencias

Enumeran los componentes que usan a éste. Se podrá indicar su naturaleza: invocación de operación, datos compartidos, inicialización, creación, etc.

5.n.6 Interfases

Describe cómo otros componentes interactúan con éste. Establecer las distintas formas de interacción y las reglas para cada una de ellas: paso de parámetros, zona común de memoria, mensajes, etc. Se indicarán las restricciones, rangos, errores, etc.

5.n.7 recursos

Describe los elementos usados por este componente que son externos a este diseño: impresoras, particiones de disco, organización de la memoria, librerías matemáticas, posibilidades de interbloqueos, etc.

5.n.8 Referencias

Todas las referencias utilizadas

5.n.9 Proceso

Algoritmos o reglas que utiliza el componente para realizar su función. Refinamiento de la subsección 5.n.3

5.n.10 Datos

Describe los datos internos del componente incluyendo al método de representación, valores iniciales, formato, valores válidos, etc. Se puede realizar mediante un diccionario de datos. Se indicará el significado de cada elemento.

6. VIABILIDAD Y RECURSOS ESTIMADOS

Analiza la viabilidad de la realización del sistema y se concretan los recursos que se necesitan para llevarlo a cabo.

7. MATRIZ REQUESITOS/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.

DOCUMENTO DDD

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

Este documento es bastante similar al ADD. La diferencia entre ambos es el nivel de detalle al que se descende. En este documento existen un mayor número de componentes y para cada uno de ellos se baja incluso hasta el nivel de codificación. Los listados fuente se recogen dentro del documento como un apéndice.

Destacar la sección 2 dedicada a recoger todas las normas, convenios y procedimientos de trabajo que se deben aplicar durante el desarrollo del sistema. Esta sección es muy importante y de ella 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.

Estructura del documento DDD :

Parte 1. DESCRIPCIÓN GENERAL

1. INTRODUCCIÓN

1.1 Objetivo

1.2 Ámbito

1.3 Definiciones, siglas y abreviaturas

1.4 Referencias

1.5 Panorámica

2. NORMAS, CONVENIOS Y PROCEDIMIENTOS

2.1 Normas de diseño de bajo nivel

2.2 Normas y convenios de documentación

2.3 Convenios de nombres (ficheros, programas, módulos, etc.)

2.4 Normas de programación

2.5 Herramientas de desarrollo de software

Parte 2. ESPECIFICACIONES DE DISEÑO DETALLADO

n. Identificador del componente

n.1 Identificador

n.2 Tipo

n.3 Objetivo

n.4 Punción

n.5 Subordinados

n.6 Dependencias

n.7 Interfases

n.8 Recursos

n.9 Referencias

n.10 Proceso

n.11 Datos

APÉNDICE A. LISTADOS FUENTE

APÉNDICE B. MATRIZ REQUISITOS/COMPONENTES

