

## UNIDAD I: INVESTIGACIÓN PRELIMINAR

### CICLO DE VIDA DE UN SISTEMA

Es el período de tiempo que transcurre desde el momento en que se decide desarrollar un software hasta el momento en que se lo entrega al usuario como producto terminado.

#### ETAPAS

1. **Estudio Preliminar o Preproyecto:** cuando se formula la solicitud comienza la primera actividad de sistemas. Muchas solicitudes que provienen de empleados y usuarios no están formuladas de manera clara; por consiguiente, antes de considerar cualquier investigación de sistemas, la solicitud del proyecto debe examinarse para determinar con precisión lo que el solicitante desea.

Un resultado importante es la determinación de que el sistema sea factible. Después de aprobar la solicitud de un proyecto se estima su costo, el tiempo necesario para terminarlo y las necesidades del personal.

2. **Análisis de Requerimientos:** estudia un dominio de información ya establecido, sin tener en cuenta cuestiones de implementación. Trata de encontrar *QUÉ* se debe hacer, no *CÓMO*. Esta etapa implica determinar las necesidades del usuario.
3. **Diseño de Sistema:** define como construir el sistema.
4. **Programación y Documentación:** el diseño debe traducirse en forma tangible para la máquina. La documentación es esencial para probar el programa y llevar a cabo el mantenimiento una vez que la aplicación se encuentra instalada.
5. **Prueba:** se centra en la lógica interna del software, asegurando que todas las sentencias se han probado, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.
6. **Implementación y Evaluación del Sistema:** proceso de verificar e instalar nuevo equipo, entrenar a los usuarios, instalar la aplicación y construir todos los archivos necesarios para utilizarla.

Dependiendo del tamaño de la organización y el riesgo asociado al uso de la aplicación, puede elegirse comenzar la operación del sistema solo en una parte de la empresa o en forma paralela con la finalidad de comparar los resultados.

7. **Mantenimiento:** aplica cada uno de los pasos precedentes del ciclo de vida a un programa existente en vez de a uno nuevo.

DOBLE V  
<http://doblev.wordpress.com>

### INVESTIGACIÓN PRELIMINAR

Comienza cuando se formula la solicitud. Esta actividad consta de:

- **Definición de los objetivos:** antes de considerar cualquier investigación de sistemas, debe examinarse la solicitud de proyecto para determinar con precisión lo que el solicitante desea (definir los objetivos).
- **Estudio de factibilidad:** la decisión sobre la factibilidad del proyecto no es una decisión del analista de sistemas, sino por la administración. Las decisiones están basadas en los datos de factibilidad recolectados en las siguientes áreas:
  - *Factibilidad técnica:* investigar si los recursos técnicos actuales pueden ser mejorados o añadidos;
  - *Factibilidad económica:* costo del estudio completo, costo de tiempo de los empleados, costo estimado de hardware y software;
  - *Factibilidad operacional:* depende de los recursos humanos disponibles para el proyecto.
- **Aprobación de la solicitud:** los proyectos que son deseables y factibles deben incorporarse en los planes de la organización.

## UNIDAD II: ANÁLISIS DE REQUERIMIENTOS

### ANÁLISIS DE REQUISITOS

Estudia un dominio de información ya establecido, sin tener en cuenta cuestiones de implementación. Trata de encontrar *QUÉ* se debe hacer, no *CÓMO*. Esta etapa implica determinar las necesidades del usuario. El análisis de requisitos permite especificar la función y el rendimiento del software, indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software. El análisis de requerimientos debe ser dirigido al que realiza las tareas (usuario, operario).

El análisis de requisitos puede dividirse en cinco áreas de esfuerzo:

- Reconocimiento del problema;
- Evaluación y síntesis;
- Modelado;
- Especificación;
- Revisión.

### PRINCIPIOS DE ANÁLISIS

Todos los métodos de análisis se relacionan por un conjunto de principios operativos:

- Debe representarse y entenderse el dominio de información de un problema;
- Deben definirse las funciones que debe realizar el software;
- Debe representarse el comportamiento del software;
- Deben dividirse los modelos que representan información, función y comportamiento de manera que se descubran los detalles por capas;
- El proceso de análisis debería ir desde la información esencial hasta el detalle de la implementación.

### MODELOS DE ANÁLISIS

Se crean para entender mejor la entidad que se va a construir. Cuando la entidad es algo físico, podemos construir un modelo a escala; pero cuando la entidad que se va a construir es un software, nuestro modelo debe ser capaz de “modelar” la información que transforma el software, las funciones y el comportamiento del sistema.

- **Modelos Funcionales:** representa una minuciosa definición de toda la funcionalidad del sistema;
- **Modelos de Comportamiento:** la característica *estímulo – respuesta* es la base de este modelo. Crea una representación de los estados del software y los acontecimientos que causan que cambie de estado.

### PROTOTIPOS

Consiste en la elaboración de un modelo o maqueta del sistema que se construye para evaluar mejor los requisitos que se desea que cumpla.

Estos modelos suelen consistir en versiones reducidas, demos o conjunto de pantallas que no son totalmente operativos. Permite intercambiar ideas con los usuarios y establecer los requerimientos.

Existen tres razones principales para emplear prototipos:

- **Prototipo de la Interfaz del Usuario:** para asegurarse de que está bien diseñada, que satisface las necesidades de quienes deben usarlo.
- **Modelos de Rendimiento:** para evaluar el posible rendimiento de un diseño técnico, especialmente en aplicaciones críticas en este aspecto.
- **Prototipo Funcional:** supone una primera versión del sistema con funcionalidades limitadas. A medida que se comprueba si las funciones implementadas son las apropiadas, se corrigen, refinan o añaden otras nuevas hasta llegar al final del sistema.

**PROCESO DE CONSTRUCCIÓN**

En base a un primer relevamiento se construye un primer sistema con los datos de entrada – salida que a nosotros nos parece. Se lo presenta al usuario y con sus comentarios y observaciones se le realizan modificaciones y se lo presenta nuevamente. Así hasta que el usuario lo aprueba.

**Ventaja:** permite obtener las necesidades del usuario.

**Desventaja:** el usuario ve el prototipo y quiere ya empezar a trabajar con el mismo, hay que hacerle entender que es un sistema vacío, que no es el definitivo.

**HERRAMIENTAS UTILIZADAS EN LA ETAPA DE ANÁLISIS****HERRAMIENTAS DEL ANÁLISIS ESTRUCTURADO**

Se deben realizar los flujos de datos a través de los distintos procesos en base a las entradas y salidas.

**DIAGRAMA DE FLUJO DE DATOS – DFD**

Es un diagrama en forma de red que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema.

- ENTES EXTERNOS:** representa un generador o consumidor de información del sistema y que no pertenece al mismo. Puede representar un subsistema, persona, departamento, organización, etc. que proporcione datos al sistema o que los reciba de él.
- PROCESOS O FUNCIONES DE TRANSFORMACIÓN:** representa una función que transforma los flujos de datos de entrada en uno o varios flujos de salida. El término proceso hay que verlo como una función que tiene que realizar el sistema.
- ALMACENES:** representa información del sistema almacenada en forma temporal. Si los flujos de datos representan datos en movimiento, los almacenes representan datos en reposo.
- FLUJO DE DATOS:** camino a través del cual viajan datos de una parte del sistema a otra. Representan datos en movimiento.

**NARRATIVA ESTRUCTURADA**

Nos permite describir procesos teniendo en cuenta las tareas que se realizan en un lenguaje sencillo; se trata de agrupar las tareas que hace el proceso.

Una vez obtenidos los datos que describen la forma en que opera el sistema, el siguiente paso a seguir es identificar los requerimientos del nuevo sistema.

**HERRAMIENTAS DE ANÁLISIS DE DECISIÓN****ÁRBOL DE DECISIÓN**

Es una representación en forma de árbol que representa los valores de las variables y las acciones tomadas para cada valor de las mismas, así como el orden en que se realiza la decisión (nº de condiciones no muy grande).

**TABLA DE DECISIÓN**

Modelo alternativo que muestra la función en forma tabular o matricial, donde en sus filas y columnas se indican condiciones y acciones. Está integrada por cuatro secciones.

Ident. de Condiciones	Entradas de Condición			
...				
...				
Ident. de Acciones	Entradas de Acciones			
...				
...				

<b>UNIDAD III: DISEÑO</b>
---------------------------

## DISEÑO

<< “Proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con suficiente detalle como para permitir su realización física” >>

Es la etapa más importante dentro del ciclo de vida de un sistema. Junto con *Codificación y Prueba* insumen el 75 % del tiempo.

El diseño de un sistema de información produce los detalles que establece la forma en que el sistema cumplirá con los requisitos identificados durante la fase de análisis. Los especialistas en sistemas se refieren a esta etapa como **diseño lógico** (el proceso de transformación que, partiendo de los requerimientos de los usuarios plasmados en el modelo lógico funcional, construye otro fácil de mantener, confiable, comprensible y con costos mínimos).

Se aplica independientemente del paradigma de desarrollo utilizado.

## TIPOS DE DISEÑO

**CONVENCIONAL**

Usada en los inicios de la computación. No seguía pasos definidos, sino que se valía del sentido común. Ligada a lenguajes de alto nivel.

**ESTRUCTURADO**

Plantea el sistema como un conjunto de funciones. El sistema está orientado a funciones, usando algún concepto de diseño descendente, desagregación. Descomposición del sistema o problema en pequeños subsistemas o subproblemas.

**ORIENTADO A OBJETOS**

Plantea el sistema como una colección de objetos que se envían mensajes entre ellos, que poseen responsabilidades y atributos.

**ORIENTADO A DATOS**

Plantea que en el sistema se deben representar los datos que se visualicen en la empresa, por mas que no se usen ahora; pero vislumbrando requerimientos posteriores. Trata de armar el sistema de acuerdo a las estructuras de datos que obtienen, es decir, analiza la entrada – salida.

## DISEÑO ESTRUCTURADO

<< “Diseño estructurado es el proceso de decidir que componentes, y la interconexión entre los mismos, para solucionar un problema bien especificado” >>

Debe ser de fácil mantenimiento. Consiste en la partición del problema en problemas más simples. Facilita el mantenimiento en etapas posteriores.

La etapa de análisis da como resultado “QUÉ” hace el sistema; la etapa de diseño es el “COMO” el sistema realiza el “QUÉ”. La calidad de un sistema está dada fundamentalmente por el diseño del mismo.

Principios (o conceptos) a tener en cuenta para obtener un buen diseño:

**1. ABSTRACCIÓN**

Es una modelización mental. Permite centrarse en un problema desde su generalización sin considerar detalles irrelevantes.

Se pueden plantear muchos niveles de abstracción. A nivel superior de abstracción, se establece una solución en términos amplios; a niveles más bajos, se toma una orientación más procedimental. Finalmente al nivel inferior, la solución se establece de manera que pueda implementarse directamente.

**2. REFINAMIENTO**

El refinamiento es un proceso de elaboración, donde se empieza con un alto nivel de abstracción.

En el refinamiento el diseñador va elaborando el enunciado original, proporcionando mas detalles con cada refinamiento sucesivo (se va mejorando el diseño que estamos realizando).

En cada parte del refinamiento se descomponen una o varias instrucciones del programa en cuestión en instrucciones más detalladas.

### 3. MODULARIDAD

Se divide al sistema en componentes identificables y tratables por separado llamados módulos, que están integrados para satisfacer los requisitos del programa. Ante un problema es más fácil identificarlo; se busca por módulos y no en todo el sistema.

Se ha dicho que “modularidad es el atributo del software que permite a un programa ser manejable intelectualmente”. Un software monolítico (compuesto por un solo módulo) no puede ser entendido fácilmente por un lector.

### 4. OCULTAMIENTO DE LA INFORMACIÓN

La ocultación implica un conjunto de módulos independientes que se comunican entre ellos solo la información necesaria para conseguir la función del software. La ocultación define y refuerza las restricciones.

Un módulo debe estar diseñado de tal manera que la información que él desde otros módulos (encapsulamiento). El principio de *ocultamiento de la información* sugiere que los módulos se han de caracterizar por decisiones de diseño que los oculten unos a otros. Los módulos deben especificarse y diseñarse de forma que la información (procedimientos y datos) contenida dentro de un módulo sea accesible a otros únicamente a través de las *interfaces* formales establecidas para cada módulo.

Proporciona sus mayores beneficios cuando se requieren modificaciones durante las pruebas o el mantenimiento.

### 5. ARQUITECTURA DEL SOFTWARE

La arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera de interactuar de estos componentes, y la estructura de los datos usados por estos componentes.

### 6. JERARQUÍA DE CONTROL

Representa la organización (frecuentemente jerárquica) de los componentes del programa (módulos) e implica una jerarquía de control. No representa aspectos procedimentales del software, tales como secuencias de procesos, o la repetición de operaciones.

### 7. ESTRUCTURA DE DATOS

Es una representación de la relación lógica entre los elementos individuales de datos. 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 las alternativas de organización, métodos de acceso, capacidad de asociación y procesamiento de la información. La organización y complejidad de la estructura de datos están limitadas solo por el ingenio del diseñador. Hay sin embargo, unas pocas estructuras clásicas de datos que forman la base para construir estructuras mas sofisticadas.

### 8. PROCEDIMIENTO DEL SOFTWARE

Define la jerarquía de control sin tener en cuenta la secuencia de procesamiento y las decisiones. El procedimiento del software se centra en los detalles de procesamiento de cada módulo individualmente.

Debe proporcionar una especificación exacta del procesamiento, incluyendo la secuencia de acontecimientos, puntos exactos de decisión, operaciones repetitivas e incluso la organización de los datos.

### 9. INDEPENDENCIA FUNCIONAL

Cada módulo realiza una sola tarea y cada módulo tiene que tener el menor número posible de comunicaciones con otros módulos. Tiene una sencilla interfaz cuando se ve desde otras partes de la estructura del programa. Evita la propagación de problemas.

---

Lo que se quiere lograr como diseño tendrá cuatro partes o Pautas:

---

- a) **DISEÑO ARQUITECTÓNICO**: define la relación entre los principales elementos estructurales del programa.
- b) **DISEÑO DE DATOS**: los objetos de datos, las relaciones definidas y el contenido detallado del diccionario de datos proporcionan la base para la actividad de diseño de datos.
- c) **DISEÑO DE INTERFAZ**: describe como se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean.
- d) **DISEÑO PROCEDIMENTAL**: descripción procedimental de los elementos del software (que es lo que hace cada uno de los módulos).

---

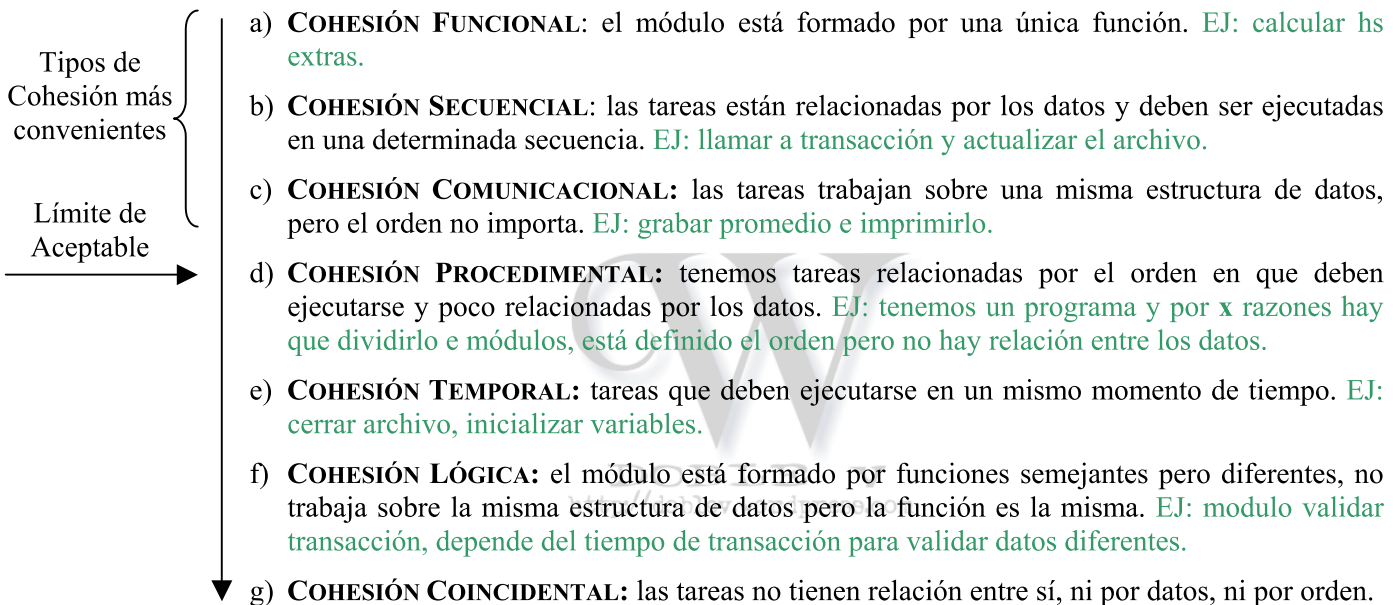
## INDEPENDENCIA FUNCIONAL

---

### CRITERIOS CUALITATIVOS (medición)

#### COHESIÓN

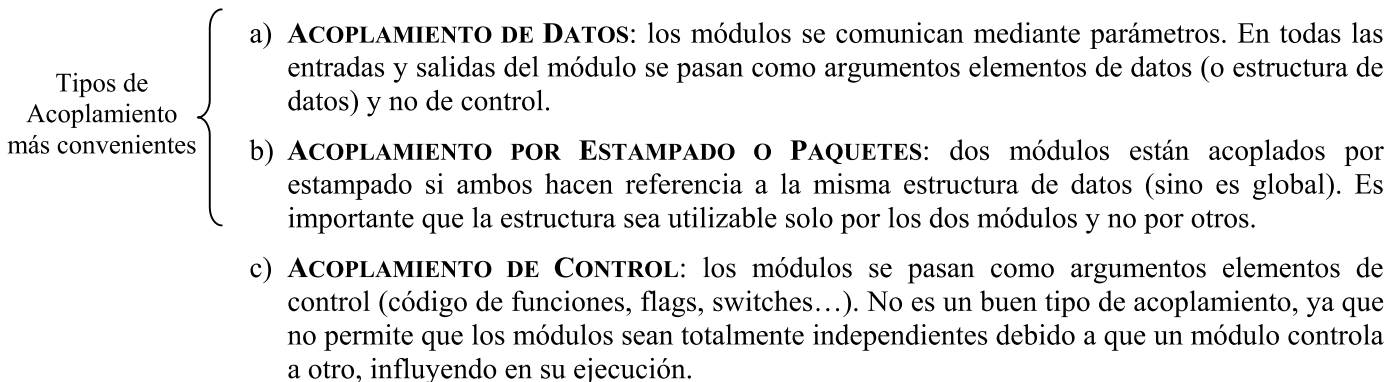
“Es una medida de la fuerza relativa funcional de un módulo”. Medida que sirve para analizar un módulo internamente; define el nivel de funcionalidad del módulo. Estudia la relación que existe entre los elementos de un mismo módulo. Separa los elementos de mayor relación de los no relacionados.



#### ACOPLAMIENTO

Se define como el grado de interdependencia entre los módulos. Para que se produzca un buen diseño, se deben hacer los módulos tan independientes como sea posible. El acoplamiento depende de la complejidad de la interfaz entre los módulos. Se intenta conseguir el menor nivel de acoplamiento posible.

#### NIVELES DE ACOPLAMIENTO



- d) **ACOPAMIENTO GLOBAL:** los módulos comparten una estructura global de datos (entorno común).

## ETAPAS DEL DISEÑO

### DISEÑO PRELIMINAR

Abarca hasta plantear una estructura arquitectónica más la definición de los datos. Consiste en el refinamiento de funciones; o sea, llegar a funciones perceptibles y bien detalladas.

El DFD se utiliza como herramienta para representar un modelo lógico. Consiste en analizar donde se justifica la incorporación de almacenes y en el refinamiento de las funciones. La incorporación de almacenes dentro de un nivel de DFD no significa refinamiento, significa detalle.

### DISEÑO DETALLADO

Incluye un refinamiento de la estructura arquitectónica, una definición detallada de la estructura de datos y una definición algorítmica de los procedimientos. Lo que se obtiene del diseño detallado es lo que entra en la etapa de codificación. Esta compuesto por:

#### a) DISEÑO DE DATOS

Maneja como vamos a definir los datos del sistema. Tiene una asociación directa con la calidad del sistema. El diseño de datos influye tanto en la estructura de los programas como en la complejidad de los procedimientos.

Traduce los objetos de datos definidos en el modelo de análisis a estructuras de datos que residen dentro del software. Consiste en identificar y definir las características de los almacenes y analizar cual es el contenido de la estructura de datos.

Aquí se hace que el modelo lógico se transforme en modelo físico. Se realiza un diccionario de datos (descripción de los datos).

#### b) DISEÑO ARQUITECTÓNICO

El objetivo es definir una estructura modular y la relación que existe entre cada uno de los módulos. Combina la estructura del programa y la estructura de datos, definiendo interfaces que permiten el flujo de datos a través del programa.

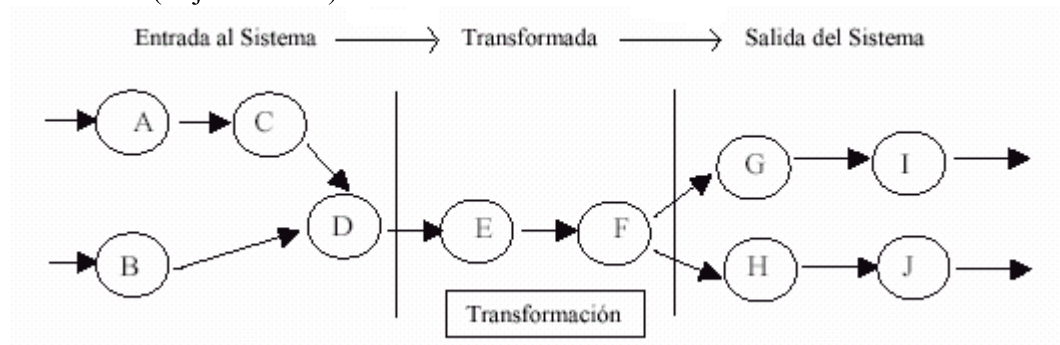
La transición desde el flujo de información a una estructura se realiza en cinco pasos:

1. Se establece el tipo de flujo de información;
2. Se indican los límites del flujo;
3. Se convierte el DFD e la estructura del programa;
4. Se define la jerarquía de control;
5. Se refina la estructura resultante usando medidas.

Existen 2 tipos de flujos de información:

#### FLUJO DE TRANSFORMACIÓN

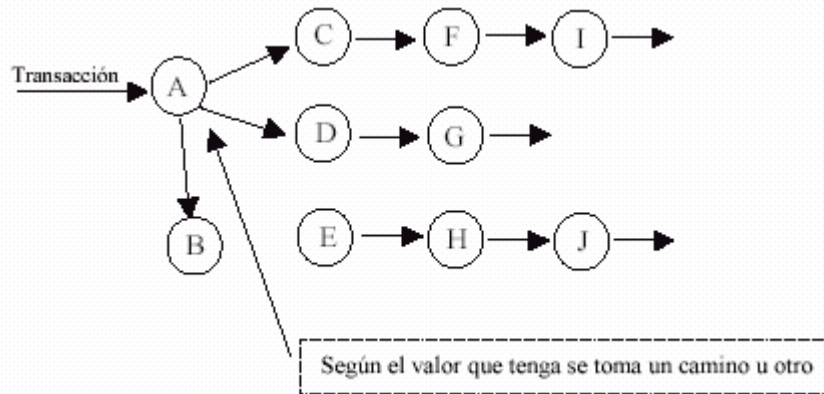
La información entra al sistema a lo largo de caminos que transforman los datos externos a un formato interno. La información entrante se pasa a través de un centro de transformación y empieza a moverse a lo largo de caminos que ahora conducen hacia “afuera” del software (flujo de salida).



### FLUJO DE TRANSICIÓN

El flujo de información está caracterizado a menudo por un único elemento de datos, denominado *transacción*; que desencadena otros flujos de información a lo largo de los muchos caminos posibles (según el valor que tenga).

Se caracteriza por los datos que se mueven a lo largo de un camino de entrada que convierte la información del mundo exterior en una transacción. La transacción se evalúa y basándose en ese valor, se inicia el flujo a lo largo de uno de los muchos caminos de acción.



### c) **DISEÑO DE INTERFAZ**

Describe como se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean. El diseño de interfaz se concentra en tres áreas importantes:

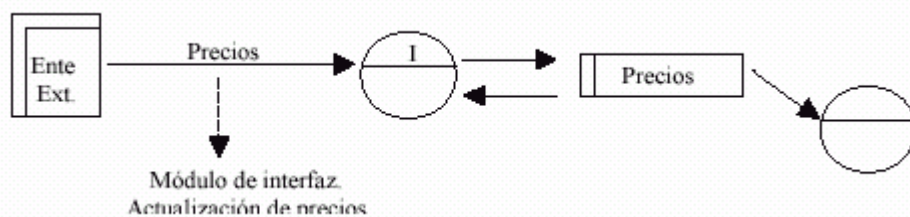
#### DISEÑO DE INTERFAZ ENTRE MÓDULOS

Lo que hay que planificar o definir es como se relacionan los módulos considerando que entre ellos fluyen datos. Se debería tener definido el lenguaje en el que se va a desarrollar el sistema.

#### DISEÑO DE INTERFAZ EXTERNA

Trata de cómo se vinculan los entes externos al sistema, con el sistema.

Se determinan los requisitos de datos y control de las entidades externas y se diseñan las apropiadas interfaces externas. Es conveniente tener almacenada la información para cuando se la necesite.



Ambos diseños de interfaces deben acoplarse con algoritmos de validación de datos y corrección de errores dentro del módulo, para evitar la propagación de efectos secundarios (que no sobrepasen los límites establecidos).

#### DISEÑO DE INTERFAZ HOMBRE-MÁQUINA

Tiene mas que ver con la parte visual del sistema. Tiene como objetivo lograr un sistema amigable y fácil de manejar.

El proceso general para diseñar la interfaz de usuario empieza con la creación de diferentes modelos de función del sistema (tal y como se percibe desde afuera). Se definen las tareas orientadas al hombre y a la máquina, requeridos para conseguir la función del sistema.

Existen cuatro aspectos a definir: TIEMPO DE RESPUESTA – FACILIDADES DE AYUDA – MENSAJES DE ERROR (entendibles e indicativos) – ESTRUCTURAS DE ÓRDENES (teclas de función para ejecutar órdenes).

#### **CRITERIOS PARA DESARROLLAR UNA INTERFAZ**

- Que sea fácil de usar;
- Que sea homogénea;
- Que sea intuitiva;
- La carga de datos debe ser rápida;
- El sistema debe ser tolerante a errores;
- Se deben realizar las acciones del sistema de modo que sea encadenado;
- La información brindada no deja lugar a dudas;
- No hay que mostrar información excesiva (innecesaria).

#### **CONSEJOS PARA REALIZAR UN SISTEMA MÁS AMIGABLE**

- Consistencia en los distintos procesos;
- Las respuestas deben ser significativas;
- Pedir verificación de cualquier acción destructiva importante;
- Permitir deshacer la mayoría de las acciones;
- Reducir la cantidad de información que debe memorizar entre acciones;
- El sistema debe autoprotgerse de errores;
- Organizar las pantallas de acuerdo a las funciones;
- Proporcionar ayudas sensibles al contexto;
- Usar verbos de acción sencillos para nombrar órdenes.

#### **d) DISEÑO PROCEDIMENTAL**

Consiste en detallar los procedimientos que forman el sistema, indicando que se debe hacer con los datos que se obtiene, que se validan, que se verifican. Sin ambigüedades.

El diseño procedimental es la especificación funcional de los módulos y requiere definir los detalles algorítmicos en un lenguaje natural. Esta especificación es una especie de programa o pseudocódigo que especifica las entradas, salidas, etc.

## UNIDAD IV: PROGRAMACIÓN

### PROGRAMACIÓN

Consiste en transformar los distintos módulos que componen el sistema en instrucciones que sean ejecutables por una computadora. El objetivo de esta etapa es producir programas confiables y de fácil mantenimiento.

En esta etapa trabaja el programador, escribe instrucciones – código. Se le debe entregar la definición de las estructuras de datos y la descripción del proceso a programar, el diseño de interfaces y los datos de prueba de los códigos.

El analista aprueba o desaprueba el código escrito, acompaña al programador de tal manera que vaya entendiendo lo que se le ha pedido. Esta etapa también se denomina diseño físico del software y los archivos.

### ESPECIFICACIÓN FUNCIONAL

Es el detalle que los analistas les entregan a los programadores por cada proceso definido. En ella consta toda la información necesaria para desarrollar el software (diseño de pantallas y de archivos, lenguaje de programación, nombres de los archivos y variables, líneas de mensajes de error, ayudas, etc.).

### OCULTACIÓN DE LA INFORMACIÓN

A cada unidad de programa solo se le debe permitir el acceso a aquellos objetos de programas que necesite para realizar su función. El acceso a otros objetos que la unidad no necesita, debe ser negado.

La ventaja de ocultar información innecesaria, es que no hay manera de que la información oculta sea corrompida por una unidad de programa que se supone no debe utilizarla.

### ESTILO DE PROGRAMACIÓN

Un programa bien escrito está bien organizado, emplea nombres significativos, incluye comentarios sensatos y utiliza construcciones del lenguaje que hacen óptimas la seguridad y legibilidad. La creación de tales programas requiere cuidado, disciplina y profesionalidad en el trabajo por parte del programador.

### INSTRUMENTOS DE SOFTWARE

Mejoran la productividad del programador, relevándolo de las tareas administrativas que antes realizaba. Algunos de ellos son: las bibliotecas de subrutinas generales, editores de línea, traductores de programa (a código máquina), instrumentos de análisis de programas, instrumentos de administración de configuración, etc.

### METODOLOGÍAS ALTERNATIVAS DE PROGRAMACIÓN

Si el programa se considera como una jerarquía de componentes:

DESARROLLO DESCENDENTE: implica empezar en el tope de la jerarquía y trabajar hacia abajo;

DESARROLLO ASCENDENTE: se empieza en el nivel mas bajo y se prosigue hacia arriba.

El desarrollo descendente es una metodología superior porque da como resultado la creación de programas más legibles y confiables que los aplicados con técnicas descendentes.

### CUESTIONES COMUNES A TODOS LOS PROGRAMAS

#### PRODUCTIVIDAD

Es una medida de la cantidad de programación o tiempo de trabajo que se realiza en un tiempo dado. Es importante porque sin una estimación de productividad es imposible la asignación de tiempo al proyecto.

#### EFICIENCIA

Resulta importante minimizar la cantidad de tiempo de CPU requerido por el programa, la utilización de la memoria, al igual que otros recursos como el disco. Mucho tiempo en el desarrollo de un programa eficiente, es probable que este sea menos mantenible y menos transportable.

**CORRECCIÓN**

Si el programa no funciona correctamente, no importa que tan eficiente sea. Se requiere que el programador declare la naturaleza de las variables y así el lenguaje pueda revisar todo cuidadosamente para evitar referencias ilegales a los datos.

**PORTABILIDAD**

Es muy importante escribir los programas de manera que se puedan aplicar a mas de una configuración del sistema de computo y del Sistema Operativo.

**MANTENIBILIDAD**

Debemos recordar que los sistemas viven durante mucho tiempo, por lo que el software debe mantenerse.

---

**EL PAPEL DEL ANALISTA**

- Debe supervisar y coordinar todas las tareas de programación que se van a realizar.
- Proporciona datos de prueba.
- Puede desarrollar manuales de usuario.
- Actúa ante consulta de los programadores.
- Puede llegar a realizar tareas de programación.
- Responde ante posibles cambios en los requerimientos de los usuarios.

**LA ELECCIÓN DEL LENGUAJE DE PROGRAMACIÓN DEPENDE:**

- Del tamaño del proyecto y el volumen de datos que maneja.
- De la organización, porque pueden tener un determinado software comprado.
- Del conocimiento de lenguajes que tienen los programadores.
- Transportabilidad del software.

## UNIDAD V: PRUEBA

### PRUEBA

Es un elemento crítico para la calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación.

Consiste en crear una serie de “*casos de prueba*” que intentan << demoler >> el software construido. Debe probarlo una persona ajena al desarrollo. Conviene probar cada módulo antes que probar el sistema completo.

#### OBJETIVOS DE LA PRUEBA

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta el momento.

La prueba no puede asegurar la ausencia de defectos, solo puede demostrar que existen defectos en el software.

### DATOS DE PRUEBA

#### REALES

Son aquellos que se extraen de los archivos de la organización. Son muy difíciles de conseguir en cantidad suficiente como para desarrollar pruebas extensas. Estos datos mostrarán como funciona el sistema para los requerimientos típicos de procesamiento (los datos reales generalmente no abarcan todas las combinaciones o formatos posibles).

#### ARTIFICIALES

Se crean solamente a fines de efectuar las pruebas. Dado que se pueden manipular todas las combinaciones de formatos y valores, harán posible las pruebas de todas las rutas lógicas y de control en todo el programa. Los programas de pruebas más exitosos utilizan lotes de prueba artificiales.

### DISEÑOS DE CASOS DE PRUEBA

El principal objetivo del diseño de casos de prueba es obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software. Para llevar a cabo este objetivo, se usan dos categorías diferentes:

#### MÉTODO DE CAJA BLANCA (enfoque estructural)

Consiste en centrarse en la estructura interna del programa para elegir los casos de prueba. La prueba ideal del software consistiría en probar todos los posibles caminos de ejecución, a través de las instrucciones de código, que puedan trazarse, esto es asegurar que por lo menos una vez, durante la prueba, se ejecutan todas las sentencias del programa y que se ejercitan todas las condiciones lógicas. Son típicamente aplicadas a pequeños componentes de programas (módulos o pequeños grupos de ellos).

- **Prueba del camino básico:** obtiene un conjunto de pruebas linealmente independiente (usa matrices de grafos) que asegura que se ejecute por lo menos una vez cada sentencia del programa.
- **Prueba de condición:** ejercita las cada una de las condiciones lógicas en el módulo de un programa lo que amplía la cobertura de la prueba y mejora la calidad de la prueba de caja blanca.
- **Prueba de flujo de datos:** selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de bucles:** se centra en la validez de las construcciones de bucles.

## MÉTODO DE CAJA NEGRA (enfoque funcional)

Consiste en estudiar la especificación de las funciones, la entrada y la salida del sistema para derivar los casos. La prueba ideal consistiría en probar todas las posibles entradas y salidas del programa.

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software y no sobre el funcionamiento interno del mismo. O sea, los casos de prueba pretenden demostrar que las funciones de software son operativas, que la entrada se acepta en forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Intenta encontrar errores en las siguientes categorías:

- Funciones incorrectas o ausentes;
- Errores de interfaz;
- Errores en estructuras de datos;
- Errores de rendimiento;
- Errores de inicialización y de terminación.

---

## TIPOS DE PRUEBA

### PRUEBA DE UNIDAD

Centra el proceso de verificación en la menor unidad del diseño del software: el módulo (cada uno en forma independiente). Se prueban los caminos de control importantes, con el fin de descubrir errores dentro del límite del módulo.

Este paso se puede llevar a cabo en paralelo para múltiples módulos. Lleva la prueba si o si de la caja blanca y se puede complementar con la caja negra.

Esta prueba se simplifica cuando existe un alto grado de cohesión.

### PRUEBA DE INTEGRACIÓN

Está ligada a la forma prevista de integrar los distintos componentes del software hasta contar con el producto global que debe entregarse. Su objetivo fundamental es la prueba de interfaces (flujo de datos) entre los módulos.

**Integración Incremental:** se combina el siguiente módulo que se debe probar con el conjunto de módulos ya probados. ASCENDENTE: se comienza por los módulos hoja;

DESCENDENTE: se comienza por los módulos raíz.

### PRUEBA DE VALIDACIÓN O ACEPTACIÓN

El objetivo que se desea lograr consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario. Es la prueba para determinar si se cumplen los requisitos de aceptación marcados por el cliente.

### PRUEBA DEL SISTEMA

Es el proceso de prueba de un sistema integrado de hardware y software para comprobar si cumplen los requisitos especificados.

La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

- **Prueba de Recuperación:** es una prueba del sistema que fuerza al fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente.
- **Prueba de Seguridad:** intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios o ilegales.
- **Prueba de Resistencia:** están diseñadas para enfrentar a los programas con situaciones anormales. Ejecuta un sistema de forma que demande recursos en cantidad, frecuencia y volúmenes no frecuentes. EJ: incrementar la frecuencia de datos de entrada; ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos.

- **Prueba de rendimiento:** está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado y se da durante todos los pasos del proceso de prueba.

Se debe asegurar el rendimiento de los módulos individuales a medida que se llevan a cabo las pruebas de caja blanca. Sin embargo, hasta que no están completamente integrados todos los elementos del sistema no se puede asegurar realmente el rendimiento del sistema.

---

## DEPURACIÓN

A diferencia de la prueba, la depuración puede considerarse un arte que surge como consecuencia de la prueba. A partir de una indicación sintomática de un problema, la actividad de depuración debe rastrear la causa del error. Se tienen siempre dos resultados:

- Se encuentra la causa, se corrige y se elimina o,
- No se encuentra la causa.

Para el último caso, la persona que realiza la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a confirmar sus sospechas y el trabajo vuelve hacia atrás a la corrección del error de una forma iterativa.



## UNIDAD VI: IMPLEMENTACIÓN DEL SISTEMA

### IMPLEMENTACIÓN

Consiste en la *puesta en marcha* del sistema o proyecto en el lugar donde va a funcionar el sistema. A partir de aquí el usuario podrá hacer uso de la información que el sistema le brinda.

En toda etapa de implementación deben desarrollarse las siguientes etapas:

#### CAPACITACIÓN DEL USUARIO

OBJETIVOS: 1 – Que el usuario conozca lo que el sistema le proporciona (que información brinda).  
2 – Que cada uno de los usuarios conozca las responsabilidades que les toca.

La capacitación tomará distintos perfiles de acuerdo al tipo de usuario a capacitar:

- Permitir al **usuario final o gerencial** conocer cual es la información que brinda el sistema.
- **Capacitación operativa:** al personal de operaciones, como recuperar el sistema si se cae o si se produce un fallo de hardware o software.
- **Usuario de nivel jerárquico:** la información que el sistema necesita y cual le brinda.

Todas las personas que tendrán uso primario o secundario deberán ser capacitadas.

#### CONVERSIÓN DEL SISTEMA

Cambio del sistema viejo por el sistema nuevo. Hay diferentes formas de proceder a la conversión:

##### CAMBIO DIRECTO

A partir de tal fecha se deja de usar el sistema viejo y se empieza a usar el sistema nuevo. Se realiza cuando el sistema nuevo es sencillo, completamente distinto al viejo o el rendimiento no es crítico.

**Ventaja:** menor costo, el usuario se ve obligado a usar el sistema nuevo aunque no le guste.

**Desventaja:** si existen problemas con el sistema nuevo no se puede volver al viejo. No permite comparar resultados. Puede haber resentimiento por parte de los usuarios.

##### CONVERSIÓN EN PARALELO

Por un tiempo se corren simultáneamente el sistema viejo y el nuevo, y se examinan los resultados; cuando se obtienen los mismos resultados se pone en uso el sistema nuevo y el antiguo se lo desecha.

**Ventaja:** permite comparar resultados y rescatar errores del sistema nuevo. Proporciona un sentido de seguridad a los usuarios.

**Desventaja:** mayor costo de funcionamiento, las salidas de uno y otro pueden diferir. Usuario resistente al cambio.

##### CONVERSIÓN GRADUAL O POR FASES

Trata de combinar las mejores características de los dos métodos anteriores sin incurrir en todos los riesgos. En este plan el número de transacciones es aumentado gradualmente conforme el sistema avanza en sus fases.

**Ventaja:** involucración gradual de los usuarios. Posibilidad de detección y recuperación de errores sin perder mucho tiempo.

**Desventaja:** lleva demasiado tiempo poner al sistema nuevo en su lugar. Inadecuado para la conversión de sistemas pequeños no complicados.

##### ENFOQUE PILOTO (MODULAR)

Se instala el sistema nuevo en una parte de la organización, si funciona se extiende a otra parte. Se usa la construcción de prototipos operacionales modulares para cambiar del sistema antiguo al nuevo en forma gradual.

**Ventaja:** errores localizados. Los usuarios se familiarizan con cada módulo.

**Desventaja:** puede llevar mucho tiempo. No siempre es posible la construcción de prototipos.

## **MIGRACIÓN DE DATOS**

Se deben convertir los datos del sistema viejo para colocarlos en el sistema nuevo. Estos pueden cambiar: formato de datos, medios de almacenamiento, contenido. Si el sistema viejo no es computarizado, se puede planificar una carga masiva o individual.

## **EVALUACIÓN O SEGUIMIENTO**

Se ha estado evaluando la evolución del sistema de información para dar retroalimentación para su mejora eventual. La evaluación también es llamada para dar continuidad a la implementación del sistema. Consiste en el control o seguimiento sobre todas las etapas de la implementación, se ve como va funcionando el sistema y se va acomodando.

### TÉCNICAS DE EVALUACIÓN:

- Análisis costo-beneficio;
- Evaluación de decisiones revisadas;
- Involucramiento del usuario;
- Enfoque de utilidad del sistema.

## **RESPALDO Y RECUPERACIÓN DE ARCHIVOS**

Para protegerse contra la pérdida de datos, los analistas diseñan procedimientos de respaldo. Se hacen copias duplicadas de los datos para garantizar que siempre existirá una copia de los registros, aún cuando el original se dañe o se destruya.

## **AUDITORIA:**

Los analistas deben tener capacidad de validar los reportes y salidas, y de probar la autenticidad y precisión de los datos e información que arroja el sistema. Entre los procesos de auditoria y control tenemos:

- Rastreo de transacciones y de análisis de valores intermedios producidos durante el procesamiento
- Impresión de registros seleccionados que cumplan ciertos criterios para validar la autenticidad de los datos
- Mantenimiento constante de las situaciones financieras cuando el sistema maneje este tipo de información
- Controles suficientes de las entradas de datos, dado que un sistema para ser confiable debe asegurar datos confiables, precisos y creíbles.

## **SEGURIDAD**

La seguridad es una necesidad ya que la información es un recurso organizacional fundamental. Aunque no existe un sistema totalmente seguro, el analista pretende disminuir la vulnerabilidad del sistema.

### SEGURIDAD FÍSICA

Seguridad de las instalaciones de computación (Hardware y Software).

### SEGURIDAD LÓGICA

Controles lógicos (contraseñas y códigos de autorización) dentro del mismo software.

### SEGURIDAD DE COMPORTAMIENTO

Se debe investigar a los empleados que eventualmente tendrán acceso a las computadoras, datos e información, para asegurarse que sus intereses sean consistentes con los de la organización y que comprendan completamente la importancia de llevar a cabo procedimientos de seguridad.

## UNIDAD VII: MANTENIMIENTO

### MANTENIMIENTO

<< “Proceso de modificar un sistema o componente software después de su entrega para corregir defectos, mejorar el rendimiento u otros atributos o adaptarlos a un entorno cambiante” >>

Consideremos el mantenimiento en función de todas las actividades que se realizan una vez entregado el producto y aceptado por el cliente.

- Corrección de defectos en el software.
- Creación de nuevas funcionalidades por nuevos requisitos de usuario.
- Mejora de la funcionalidad y del rendimiento.

### TIPOS DE MANTENIMIENTO

#### MANTENIMIENTO PERFECTIVO

Actividades para mejorar o ~~añadir~~ nuevas funcionalidades requeridas por el usuario y/o el analista. EJ: mejorar listados o gráficos.

#### MANTENIMIENTO ADAPTATIVO

Actividades que se realizan para adaptar el sistema a los cambios en su entorno. No se realizan mantenimientos por cambios de datos como porcentajes aplicados, ya que estos son considerados como pautas. EJ: bono federal – legislación. Los cambios pueden ocurrir en:

- **Entorno de datos:** EJ: pasar de un sistema de archivos a uno relacional (base de datos).
- **Entorno de proceso:** nueva plataforma de explotación o nuevo sistema operativo.

#### MANTENIMIENTO CORRECTIVO

Actividades dedicadas a corregir defectos en el hardware o en el software detectados por el usuario durante la explotación del sistema.

- **Procesamiento:** EJ: salidas incorrectas.
- **Rendimiento:** EJ: tiempos de respuesta debajo de lo estimado.
- **Implementación:** EJ: inconsistencia en el diseño.com

#### MANTENIMIENTO PREVENTIVO

Actividades para facilitar el mantenimiento futuro del sistema. EJ: incluir validación de los datos de entrada. Depuración de la base de datos.

### MANTENIMIENTO DE LA DOCUMENTACIÓN

A medida que se va modificando un sistema de programación, la documentación asociada también se debe modificar para que refleje los cambios en el sistema. Si el cambio es transparente a los usuarios, solo es necesario modificar aquellos documentos que describen la aplicación del sistema.

## UNIDAD VIII: OTRAS METODOLOGÍAS

### ORIENTACIÓN A OBJETOS

Sirve para solucionar las falencias del diseño estructurado y para hacer una aproximación a la realidad. En el mundo real no se ven funciones, se ven objetos.

**OBJETO:** Un objeto es la instancia de una clase. Es cualquier cosa distinguible o identificable, con determinados atributos y determinado comportamiento.

**ABSTRACCIÓN:** Examen selectivo de ciertos aspectos del problema, eliminando los aspectos que no son importantes. La abstracción debe servir para un propósito que nos determina lo que no es importante, delimitando nuestro universo. Es posible obtener múltiples abstracciones de la misma cosa. Un buen modelo captura los aspectos cruciales del problema y omite el resto.

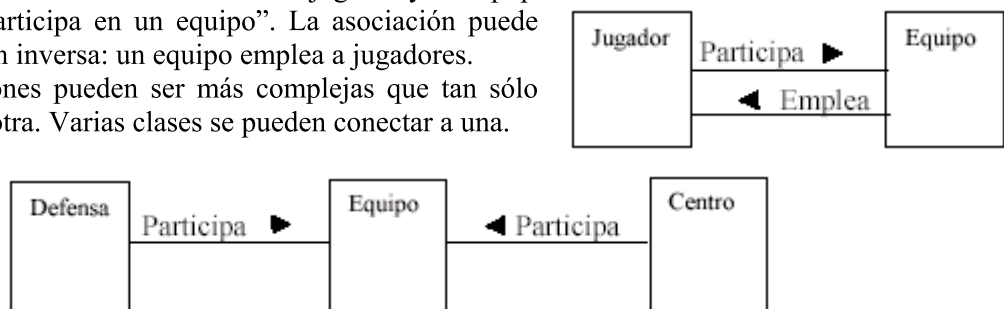
**HERENCIA:** Es la capacidad de una clase para definir el comportamiento y las estructuras de datos de sus ejemplos como un súper conjunto de la definición de otra clase o clases. Permite concebir una nueva clase de objetos como un refinamiento de otra, para abstraer las similitudes entre las clases, y para diseñar y especificar únicamente las diferencias para la clase nueva. Únicamente las diferencias para la clase nueva. Las clases pueden heredar comportamiento.

**POLIMORFISMO:** Es la capacidad de dos o más clases de objetos para responder al mismo mensaje, cada una de su manera propia. Esto significa que un objeto no necesita saber a quién envía un mensaje. Sólo necesita saber que muchos tipos diferentes de objetos se han definido para responder a ese mensaje en particular. Por ejemplo, la acción *abrir* realizará operaciones diferentes en el caso que sea enviado este mensaje a un objeto ventana, puerta, periódico, un regalo, etc. Cada clase sabe como realizar tal operación.

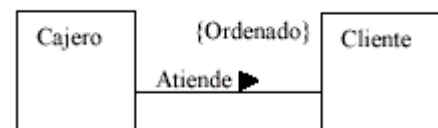
**ENCAPSULAMIENTO:** La esencia del encapsulamiento es que cuando un objeto trae consigo una funcionalidad, esta última se oculta. Esto permite reducir el potencial de errores que pudieran ocurrir. Públicamente se conoce de objeto sus capacidades: “Yo puedo hacer estas cosas”, y sus salidas: “Yo puedo decir estas cosas”. Pero no se conoce cómo sabe o cómo lo hace. El objeto que requiere información, especifica el trabajo o pide la información y lo deja. Luego de decidir qué funcionalidad e información utilizarán los otros objetos se oculta el resto. Se diseña una interfaz pública (la parte de afuera de la cápsula) que permite a otros objetos acceder a lo que ellos requieren. La representación privada (lo de adentro de la cápsula) se protege del acceso de otros objetos.

**ASOCIACIONES:** Cuando las clases se conectan entre sí de forma conceptual, esta conexión se conoce como asociación. Por ejemplo: la asociación entre un jugador y un equipo. Podrá caracterizar tal asociación con la frase: “un jugador participa en un equipo”. La asociación puede funcionar en dirección inversa: un equipo emplea a jugadores.

Las asociaciones pueden ser más complejas que tan sólo una clase asociada a otra. Varias clases se pueden conectar a una.

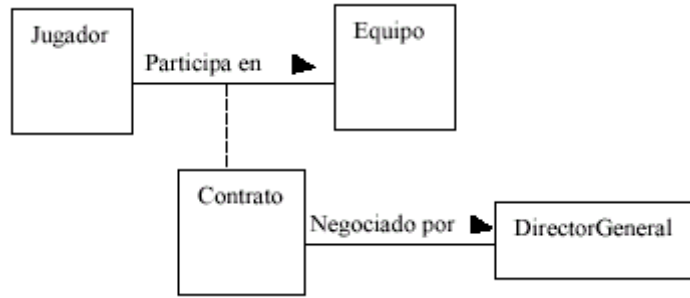


**RESTRICCIONES:** Puede establecerse una restricción en las asociaciones. En el ejemplo siguiente: la asociación atiende está restringida para que el cajero atienda al cliente en turno.



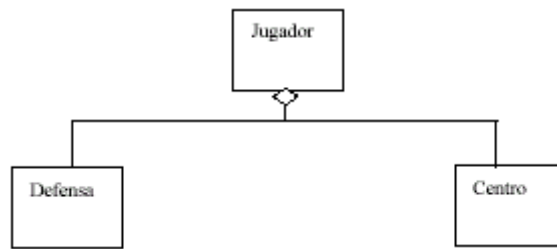
**CLASES DE ASOCIACIÓN:** Una asociación, al igual que una clase, puede contener atributos y operaciones. De hecho cuando éste sea el caso, tendremos una clase de asociación. Una clase de asociación modela los atributos y operaciones de una asociación. Se conecta a una asociación mediante una línea discontinua, y puede asociarse a otra clase.

Un **vínculo** es la instancia de asociación. Conecta a los objetos en lugar de las clases.

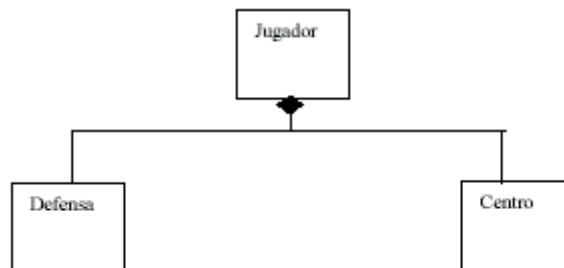


**MULTIPLICIDAD:** La multiplicidad señala la cantidad de objetos de una clase asociada. Cuando la multiplicidad de una asociación es de uno a muchos, con frecuencia se presenta un reto muy particular: la búsqueda. Cuando un objeto de una clase tiene que seleccionar un objeto particular de otro tipo para cumplir con un papel en la asociación, la primera clase deberá atenerse a un atributo en particular para localizar al objeto adecuado. En ocasiones, una clase es una asociación consigo misma (asociación reflexiva). Esto puede ocurrir cuando una clase tiene objetos que pueden jugar diversos papeles. Por ejemplo: un ocupante de automóvil puede ser un conductor o un pasajero.

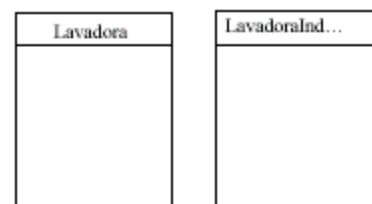
**AGREGACIÓN:** En ocasiones una clase consta de otras clases. Éste es un tipo especial de relación conocida como agregación. Puede establecerse una restricción a una agregación para mostrar que un componente u otro es parte del todo.



**COMPOSICIÓN:** Una composición es un tipo muy representativo de una agregación. Cada componente dentro de una composición puede pertenecer tan sólo a un todo. Los componentes de una mesa de café (la superficie de la mesa y las patas) establecen una composición.

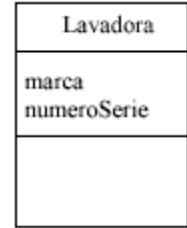


**CLASES:** Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio. Los objetos que comparten el mismo comportamiento se dice que pertenecen a la misma clase. Una clase es una especificación genérica para un número arbitrario de objetos similares. Un rectángulo es el símbolo que representa una clase. El nombre de la clase es, por convención, una palabra con la primera letra en mayúscula y normalmente se coloca en la parte superior del rectángulo. Si el nombre de la clase consta de dos palabras entonces, debemos unir las e iniciar cada una con mayúsculas. Gráficamente:



**ATRIBUTO:** Un atributo es una propiedad o característica de una clase y describe un rango de valores que la propiedad podrá contener en los objetos de la clase. Describen de alguna manera la clase o el objeto. Están asociados a clases y objetos; puede tomar un valor definido por un dominio enumerado.

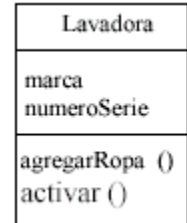
Una clase podrá contener varios o ningún atributo. Por convención, si el atributo consta de una sola palabra se escribe en minúsculas; por otro lado, si el nombre contiene más de una palabra, cada palabra será unida a la anterior y comenzará con una letra mayúscula, a excepción de la primer palabra que comenzará en minúscula.



Gráficamente:

El UML da la opción de indicar información adicional de los atributos. En el símbolo de la clase, se podrá especifica un tipo para cada valor del atributo, así como también un valor predeterminado. Por ejemplo:

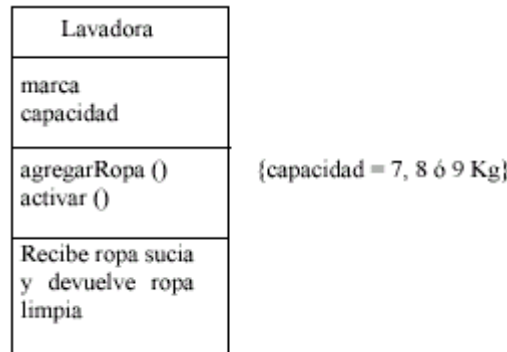
marca: string = "Philips"  
capacidad: integer



**OPERACIONES:** En UML no se hace referencia a métodos sino a operaciones. Una operación es algo que la clase puede realizar u otra clase puede hacer a una clase. De la misma manera que el nombre de un atributo, el nombre de una operación se escribe en minúsculas si consta de una sola palabra. Gráficamente:

Así como es posible establecer información adicional de los atributos, también lo es en lo concerniente a las operaciones. En los paréntesis que preceden al nombre de la operación se podrá mostrar el parámetro con el que funcionará la operación junto con su tipo de dato.

**RESPONSABILIDADES Y RESTRICCIONES:** La responsabilidad es una descripción de lo que hará la clase, es decir, lo que sus atributos y operaciones intentan realizar en conjunto. Una lavadora, por ejemplo, tiene la responsabilidad de recibir ropa sucia y dar por resultado ropa limpia. En el símbolo, indicará las responsabilidades en un área inferior a la que contiene las operaciones. Una manera más formal es agregar una restricción, un texto libre bordeado por llaves. Por ejemplo, supongamos que en la clase Lavadora deseamos establecer la capacidad de una lavadora será de 7, 8 o 9 (y así restringir el atributo capacidad de la clase Lavadora)



**MENSAJE:** Un objeto accede a otro objeto enviándole un mensaje. Un mensaje consiste del nombre de una operación y cualquier argumento requeridos. El conjunto de mensajes al que un objeto puede responder es conocido como el comporta- miento del objeto. Un objeto puede enviar mensajes privados a sí mismo para implementar operaciones públicamente accesibles.

**INSTANCIA:** Los objetos que se comportan en una manera especificada por una clase se llaman las instancias de esa clase. Todos los objetos son las instancias de alguna clase. Una vez que una instancia de una clase se crea, se comporta como todas las otras instancias de su clase, y es capaz de recibir un mensaje para desempeñar cualquier operación para la que tiene métodos.

**SUBCLASE:** Es una clase que hereda el comportamiento desde otra clase. Una subclase comúnmente agrega su comportamiento propio para definir su tipo único de objeto.

**SUPERCLASE:** Una superclase es una clase cuyo comportamiento específico se hereda. Una clase podría tener una única superclase, o podría tener varias combinando comportamiento desde varias fuentes y agregar solo un poco e si misma para producir su único tipo de objeto.

**CLASE ABSTRACTA:** No son destinadas para producir instancias de sí mismas. Existen para que el comportamiento común a una variedad de clases pueda factorizarse en una ubicación común, donde puede definirse una vez y reutilizarlo nuevamente y nuevamente. Especifican totalmente su comportamiento, pero no necesitan implementarse completamente.

---

## INTRODUCCIÓN A UML.

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. El Lenguaje Unificado de Modelado determina un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

***Permite reflejar un determinado modelo de objetos. No es una metodología, es un estándar. Es una serie de diagramas que permiten documentar todo.***

UML ofrece nueve diagramas:

- Diagramas de Casos de Uso para modelar los procesos.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

---

## DIAGRAMAS UML

### DIAGRAMA DE CLASES

Esta formado por varios rectángulos conectados por líneas que muestran la manera en que las clases se relacionan entre si. Un rectángulo es el símbolo que representa a la clase, y se divide en tres áreas:

- El área superior contiene el nombre;
- El área central contiene los atributos;
- El área inferior contiene las acciones u operaciones.

El diagrama de clases se utiliza para modelar la estructura estática de las clases del sistema.

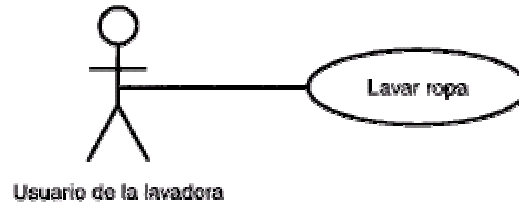


## DIAGRAMA DE CASOS DE USO

Describe la forma en que un sistema lucirá para los usuarios potenciales. Son todas las funciones que va a realizar el sistema. Es una colección de escenarios iniciados por una entidad llamada actor (persona, componente, sistema).

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Para los desarrolladores de sistema esta es una herramienta valiosa, ya que es una técnica de aciertos y errores para obtener los requerimientos del sistema desde el punto de vista del usuario.

Se utiliza para modelar los procesos que realiza el sistema.



## DIAGRAMA DE SECUENCIA

Los diagramas de clases y los de objetos representan información estática. En un sistema funcional los objetos interactúan entre sí, y tales interacciones suceden con el tiempo. El diagrama de secuencias muestra la mecánica de la interacción con base en tiempos.

Tiene dos dimensiones. La horizontal es la disposición de los objetos, y la dimensión vertical muestra el paso del tiempo.

Se utiliza para modelar el paso de mensajes entre objetos.

Un mensaje simple es la transferencia de control de un objeto a otro. Sincrónico, esperará la respuesta a tal mensaje antes de continuar. Asíncrono, no esperará respuesta para continuar.

