

UNIDAD 1:

Desarrollo de Software

Entornos de Desarrollo

1º Desarrollo de Aplicaciones Web

Antonio Vázquez Carreño

Tabla de contenido

SOFTWARE	4
Concepto	4
Definiciones de Software	4
Clasificación de software por tipo de tarea que realiza	4
Clasificación de software por método de distribución	4
Licencias de Software	5
Software libre	5
Software propietario	5
Software de dominio público	5
GPL (GNU General Public License)	6
CICLO DE VIDA DEL SOFTWARE	7
Concepto	7
Etapas	7
Modelos.....	7
En cascada	8
Evolutivo.....	9
FASES DEL DESARROLLO DE UNA APLICACIÓN	12
Resumen.....	12
Análisis.....	12
Técnicas de comunicación.....	12
Requisitos	12
Representación de los requisitos	13
Documentación de la fase de Análisis	17
Diseño.....	17
Diseño estructurado (clásico).....	18
Diseño orientado a objetos	25
Codificación	25
Normas (buenas prácticas).....	25
Programas	26
Lenguajes de programación	26
Pruebas.....	28
Casos de prueba y recomendaciones.....	28
Documentación	29

Objetivos	29
Clasificación.....	29
Estructura del documento.....	30
Explotación.....	30
Tareas	31
Mantenimiento	31
Tipos	31

SOFTWARE

Concepto

Un equipo informático se compone de dos partes:

- **Hardware:** Componentes físicos, “lo que se puede ver y tocar”, monitor, teclado, ratón, placas,...
- **Software:** Parte lógica. Comprende el conjunto de programas y aplicaciones que actúan sobre el hardware del ordenador y facilitan al usuario la realización de diferentes tareas.

Definiciones de Software

- **R.A.E.:** Es el conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.
- **Estándar 729 del IEEE:** Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Clasificación de software por tipo de tarea que realiza

- **Software de sistema:** Aquel que permite que el hardware funcione. Lo forman los programas que permiten la administración de la parte física del ordenador, y es el que interactúa entre el usuario y los componentes hardware del ordenador.
 - Sistemas operativos, controladores de dispositivos, herramientas de diagnóstico, de corrección y optimización,...
- **Software de aplicación:** Lo forman los programas que nos ayudan a realizar tareas específicas en cualquier campo susceptible de ser automatizado o asistido. Este software hace que el ordenador sea una herramienta útil para el usuario.
 - Aplicaciones ofimáticas, software educativo, software médico, aplicaciones de contabilidad, de diseño asistido, aplicaciones de control y automatización industrial,...
 - **Software multimedia:** Se refiere a los programas utilizados para presentar de forma integrada textos, gráficos, sonidos y animaciones. Ej. Enciclopedias multimedia.
- **Software de programación o desarrollo:** Es el que proporciona al programador herramientas para ayudarle a escribir programas informáticos y a usar diferentes lenguajes de programación de forma práctica.
 - Entre los que se encuentran los Entornos de Desarrollo Integrados (IDE)= Normalmente en entorno visual de forma que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar,...

Clasificación de software por método de distribución

- **Shareware:** Es una modalidad de distribución de software para que el usuario pueda evaluar de forma gratuita el producto por un tiempo determinado. Para obtener la licencia de uso completo se requiere un pago.

- **Freeware (no es igual a Software Libre):** Es un software que se distribuye sin cargo. Suele incluir una licencia de uso que permite redistribución, pero con restricciones:
 - No modificar
 - No vender
 - Dar cuenta del autor...
- **Adware:** Suelen ser programas shareware que de forma automática descargan publicidad en nuestro ordenador cuando lo instalamos o ejecutamos. Al comprar la licencia se elimina la publicidad.
- **Software diseñado para uso específico:** Este tipo de software es el que se desarrolla especialmente para resolver un problema determinado de alguna organización o persona; este software requiere de un experto en informática para su creación o adaptación.
- **Programas “enlatados”:** Software desarrollado por compañías y vendido principalmente por distribuidores. Suelen resolver necesidades comunes.

Licencias de Software

Es un contrato que se establece entre el desarrollador de un software y el usuario. Sometido a propiedad intelectual y a derechos de autor. En el cual se definen con precisión los derechos y deberes de ambas partes.

El desarrollador elige el tipo de licencia según la cual se distribuye el software.

Software libre

El autor cede una serie de libertades básicas al usuario:

- Libertad de utilizar el programa con cualquier fin en cuantos ordenadores desee.
- Libertad de estudiar cómo funciona el programa y de adaptar su código a sus necesidades específicas (código fuente).
- Libertad de distribuir copias a otros usuarios (con o sin modificaciones).
- Libertad de mejorar el programa y de hacer públicas y distribuir al público las modificaciones.

Software propietario

Se distribuye en formato binario (ejecutable), no se tiene acceso al código fuente.

Prohibiciones:

- Redistribución
- Modificación
- Copia
- Uso en varias máquinas simultáneamente
- Transferencia de titularidad
- Difusión de fallos y errores

Software de dominio público

Carece de licencia o no hay forma de determinarla. No se conoce al autor por abandono o por expiración de licencia. Todo el mundo lo puede utilizar.

GPL (GNU General Public License)

Es la más utilizada en software libre y de fuentes abiertas.

Da derecho a usar y modificar el programa con la obligación de hacer públicas las versiones modificadas de este.

Por ejemplo:

Licencia de Moodle (<https://docs.moodle.org/all/es/Licencia>)



Más información sobre licencias:

<http://www.gnu.org/licenses/license-list.es.html#SoftwareLicenses>

CICLO DE VIDA DEL SOFTWARE

Concepto

Se define como un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso. (Estándar ISO/IEC 12207-1)

Etapas

Se divide en siete etapas dentro de las cuales se realizarán diferentes tareas:

- **Análisis:** Construye un modelo de requisitos. En esta etapa se debe entender y comprender de forma detallada el problema que se va a resolver. Es muy importante producir en esta etapa una documentación entendible, completa y fácil de verificar y modificar.
 - Responde a la pregunta **¿QUÉ?**
- **Diseño:** En esta etapa ya sabemos QUÉ hay que hacer, ahora hay que definir CÓMO se va a resolver el problema. Se deducen estructuras de datos, la arquitectura del software, la interfaz de usuario y los procedimientos. Por ejemplo, en esta etapa hay que seleccionar el lenguaje de programación, el sistema gestor de bases de datos,...
 - Responde a la pregunta **¿CÓMO?**
- **Codificación:** Se traduce lo descrito en el diseño a una forma legible por la máquina. La salida de esta fase será código ejecutable.
- **Pruebas:** Se comprueba que se cumplen los criterios de corrección y calidad. Las pruebas deben garantizar el correcto funcionamiento del sistema.
- **Explotación:** Una vez que se han realizado todas las pruebas y documentado todas las etapas, se pasa a explotación. En esta etapa se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente.
- **Mantenimiento:** Esta fase tiene lugar después de la entrega del software al cliente. En ella hay que asegurar que el sistema puede adaptarse a los cambios que pueden estar provocados por:
 - Errores
 - Necesidad de adaptarse al entorno
 - Cliente requiere mejoras funcionales
- **Documentación:** Cada etapa tiene como entrada uno o varios documentos procedentes de la etapa anterior y produce otros documentos de salida.

Modelos

Es importante tener en cuenta las características del proyecto software para elegir el modelo de ciclo de vida.

Los modelos más importantes son:

- En cascada
- Evolutivo

En cascada

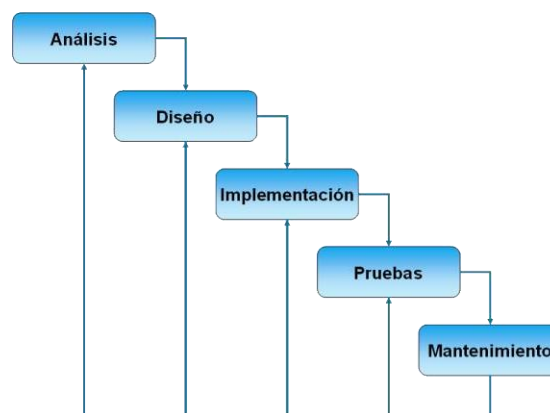
Las etapas para el desarrollo del software tienen un orden. Para empezar una etapa es necesario finalizar la etapa anterior; después de cada etapa se realiza una revisión para comprobar si se puede pasar a la siguiente.



Variante

Una **variante** de este modelo sería el **modelo en cascada con realimentación**:

Durante el desarrollo de una etapa se detectan fallos, entonces será necesario retornar a la etapa anterior, realizar los ajustes pertinentes y continuar de nuevo.



Ventajas

- Fácil de comprender
- La calidad del producto resultante es alta
- Permite trabajar con personal poco cualificado

Inconvenientes

- La necesidad de tener todos los requisitos definidos desde el principio (no siempre ocurre)
- Es difícil volver atrás si se cometen errores en una etapa.
- El producto no está disponible para su uso hasta que no está completamente terminado.

Se recomienda cuando...

- El proyecto es similar a alguno ya realizado con éxito anteriormente.
- Los requisitos son estables y están bien comprendidos.
- Los clientes no necesitan versiones intermedias.

Evolutivo

El software evoluciona con el tiempo, por lo que los requisitos del usuario y del producto cambian.

Por competencia, hay empresas que no pueden esperar a tener un producto totalmente completo, en su lugar, van introduciendo versiones cada vez más completas hasta llegar al producto final deseado.

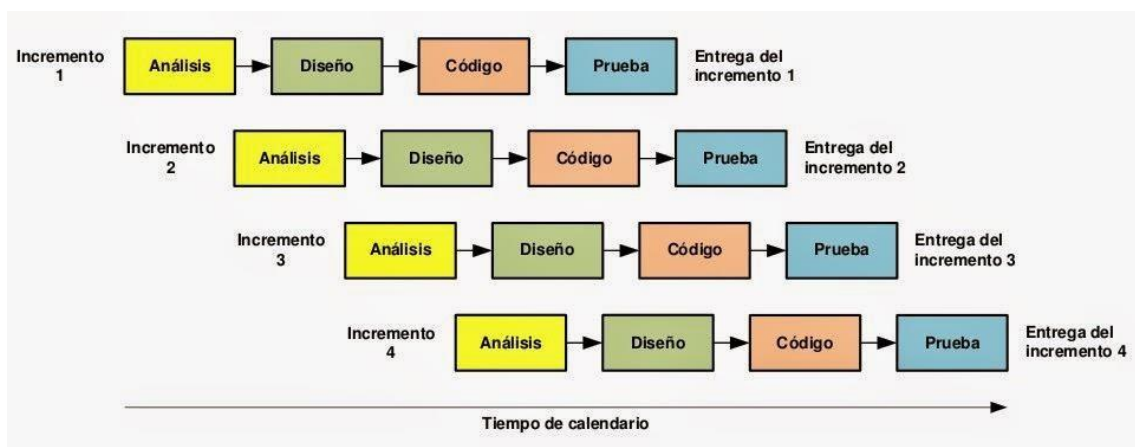
Los modelos evolutivos más conocidos son:

- Modelo iterativo incremental.
- Modelo en espiral.

Iterativo incremental

Está basado en varios ciclos en cascada realimentados y aplicados repetidamente.

El modelo incremental entrega el software el software en partes pequeñas, pero utilizables, llamadas **“incrementos”**.



Ejemplo:

Procesador de textos

- En el primer incremento se desarrollan funciones básicas de gestión de archivos y de producción de documentos
- En el segundo, se desarrollan funciones gramaticales y de corrección ortográfica
- En el tercero, se desarrollan funciones avanzadas de paginación
- Y así, sucesivamente.

Ventajas:

- No se necesitan conocer todos los requisitos al comienzo.
- Permite la entrega temprana al cliente de partes operativas del software.
- Las entregas facilitan la realimentación de los próximos entregables.
- Reduce riesgos ya que se construye a partir de un diseño preliminar, que va siendo completado con nuevos incrementos conforme el cliente va teniendo nuevos requisitos.

Inconvenientes:

- Es difícil estimar el esfuerzo y el coste final necesario.
- Se corre el riesgo de no acabar nunca.
- No recomendable para desarrollo de sistemas en tiempo real, de alto nivel de seguridad, y/o de alto índice de riesgos.

Se recomienda cuando...

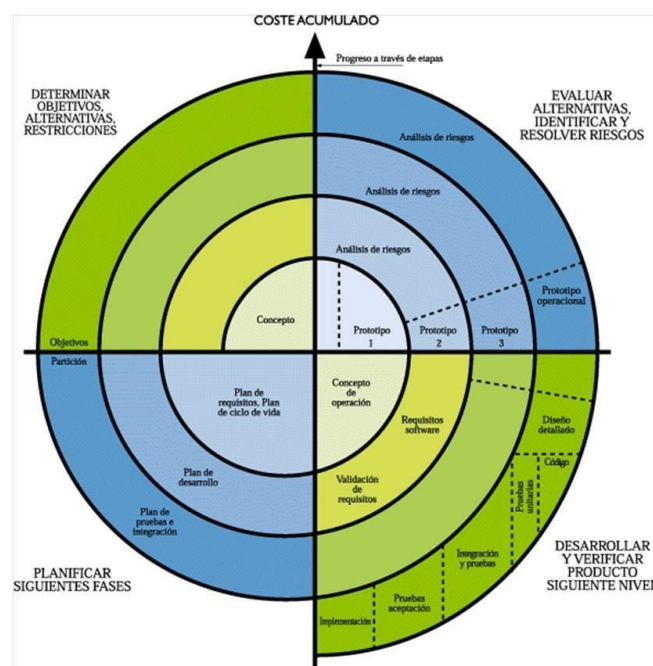
- Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.
- Se están probando o introduciendo nuevas tecnologías.

En espiral

Se representa como una espiral, donde en cada ciclo se desarrolla una parte del modelo.

Cada ciclo está formado por cuatro fases y cuando termina produce una versión incremental del software con respecto al ciclo anterior.

Parecido al iterativo incremental con la diferencia de que en cada ciclo se tiene en cuenta el análisis de riesgos.



Fases para cada ciclo:

- **Determinar objetivos:** Cada ciclo comienza con la identificación de los objetivos, las alternativas para alcanzar los objetivos (diseño a o b, reutilización, compra,...) y las restricciones impuestas a la aplicación de las alternativas (costes, plazos, interfaz,...)
- **Análisis de riesgo:** Evalúa las alternativas en relación con los objetivos y limitaciones. Se identifican los riesgos involucrados y la manera de resolverlos. (Riesgo = requisitos mal comprendidos, mal diseño, errores,...)
- **Desarrollar y probar:** Desarrollar la solución al problema en este ciclo y verificar que es aceptable.
- **Planificación:** Revisar y evaluar todo lo que se ha hecho, decidir si se continúa, y, en caso necesario planificar las fases del ciclo siguiente.

Ventajas:

- No requiere una definición completa de los requisitos para empezar a funcionar.
- Análisis del riesgo en todas las etapas.
- Reduce riesgos del proyecto.
- Incorpora objetivos de calidad.

Inconvenientes:

- Es difícil evaluar los riesgos.
- El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
- El éxito del proyecto depende en gran medida de la fase de análisis de riesgos.

Se recomienda cuando...

- Proyectos de gran tamaño y que necesitan constantes cambios.
- Proyectos donde sea importante el factor riesgo.

FASES DEL DESARROLLO DE UNA APLICACIÓN

Resumen

- **Análisis:** Se especifican los requisitos funcionales y no funcionales del sistema.
- **Diseño:** Se deducen estructuras de datos, la arquitectura del software, la interfaz de usuario y los procedimientos.
- **Codificación:** Se traduce lo descrito en el diseño a una forma legible por la máquina.
- **Pruebas:** Se prueban los programas para detectar errores y se depuran.
- **Documentación:** En todas las etapas, se documenta y guarda toda la información.
- **Explotación:** Instalamos, configuramos y probamos la aplicación en los equipos del cliente.
- **Mantenimiento:** Se mantiene el contacto para actualizar y modificar la aplicación en el futuro.

Análisis

- Trata de entender y comprender el problema y darle solución.
- Se analizan los requisitos que el sistema debe cumplir.
- La obtención de requisitos no es fácil
- Debe existir una excelente comunicación entre cliente y analistas/desarrolladores.

Técnicas de comunicación

- **Entrevistas:** Técnica más tradicional. Consiste en hablar con el cliente ¿psicología?
- **Desarrollo conjunto de aplicaciones (JAD, Joint Application Development):** Un equipo formado por usuario, administradores, analistas, desarrolladores,... se reúne en una sesión intensiva para recopilar datos, debatir ideas, conciliar las diferencias, identificar y priorizar las necesidades, y generar soluciones alternativas.
- **Planificación conjunta de requisitos (JRP, Joint Requirements Planning):** Están dirigidas a la alta dirección, se obtienen los requisitos de alto nivel o estratégicos.
- **Brainstorming:** Reuniones en grupo cuyo objetivo es generar ideas desde diferentes puntos de vista para la resolución de problemas.
- **Prototipos:** Es una versión inicial del sistema. Clarifica puntos, demuestra conceptos,...
- **Casos de uso:** Escenarios que describen cómo se usa el software en una determinada situación.

Requisitos

Se especifican dos tipos de requisitos:

- **Requisitos funcionales:** Describen con detalle la función que realiza el sistema, cómo reacciona, cómo se comporta,...
- **Requisitos no funcionales:** Tratan sobre las características del sistema, cómo puede ser la fiabilidad, mantenibilidad, sistema operativo, hardware, restricciones, limitaciones,...

Representación de los requisitos

Para representar los requisitos obtenidos haremos uso de diferentes diagramas:

- Diagramas de flujo de datos (DFD)
- Diagramas de flujo de control (DFC)
- Diagramas de transición de estados (DTE)
- Diagramas Entidad/Relación (DER)
- Diccionario de datos (DD)

DFD

Representa el flujo de datos entre los distintos procesos, entidades externas y almacenes que forman el sistema.

- **Procesos:** Identifican funciones (óvalos o círculos)
- **Entidades externas:** Representan componentes que no forman parte del sistema, pero le proporcionan datos o los reciben de él. Personas, departamentos,... (rectángulos).
- **Almacenes:** Representan el lugar donde se almacenan los datos procesados o desde donde se recuperan (dos líneas horizontales y paralelas)
- **Flujos de datos:** Representan el movimiento de datos dentro del sistema (flechas)

(Notación Yourdon)

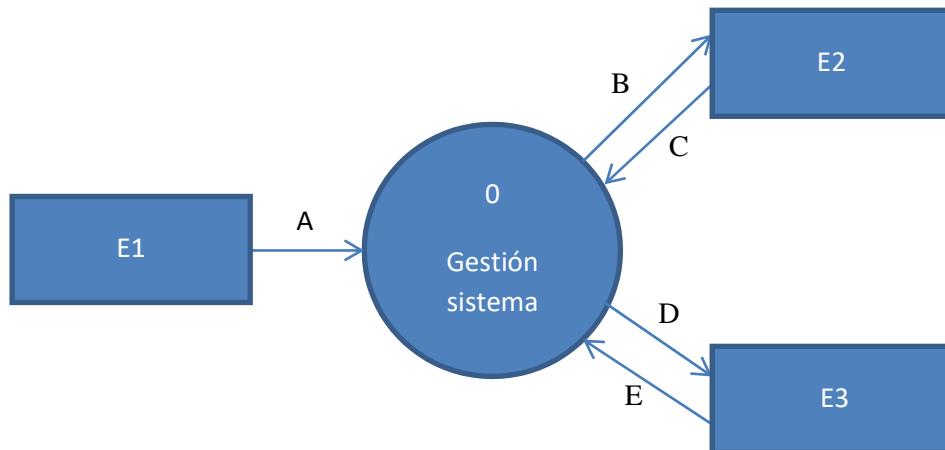
Reglas básicas:

- Los **elementos** del DFD tienen que tener un **nombre con significado** (en los flujos de datos que entran y salen de un almacén no hace falta indicar un nombre)
- Los **flujos** de datos deben mostrar su **sentido**.
- **Se permiten flujos** entre: dos procesos, un proceso y un almacén, un proceso y una entidad externa.
- **No se permiten flujos** entre: dos almacenes, dos entidades externas, un almacén y una entidad externa.
- Realizar **un DFD para cada parte del sistema** que se haya identificado.
- Los almacenes y entidades externas se pueden representar varias veces si mejora la **legibilidad**.
- No puede haber **elementos aislados**.

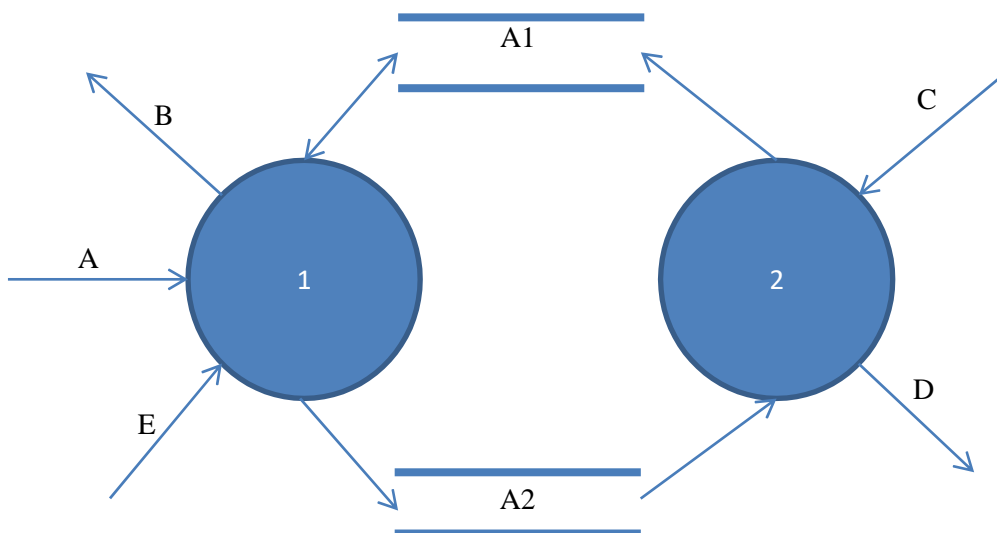
Niveles

Para representar un sistema grande se utilizan varios DFDs siguiendo una estructura jerárquica:

- **Nivel 0 o diagrama de contexto:** Representa un único proceso, identificado con un 0, que representa al sistema completo.
 - Se representa: El proceso, los flujos de entrada y salida de datos y las entidades externas.
 - No se representan: Los almacenes

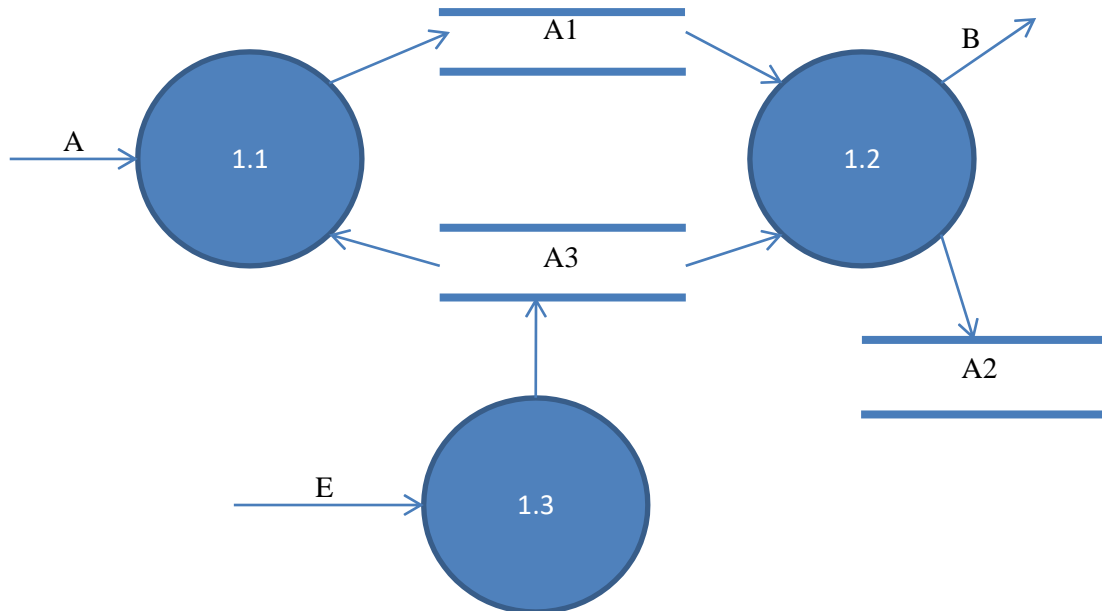


- **Nivel 1:** Se descompone el proceso identificado con un 0 en otro DFD en el que se representan las funciones principales del sistema.
 - Los procesos que aparecen se enumeran del 1 en adelante.
 - Se pueden observar los flujos de entrada y salida procedentes de las entidades externas.
 - No se representan las entidades externas, aunque a veces es conveniente para ayudar a la comprensión.



- **Nivel 2:** Se descompone cada uno de los procesos en nuevos procesos que representan funciones más simples.
 - Vamos de los general al detalle, pasando por niveles intermedios (*top-down*)

- En el DFD de nivel 2 de un proceso, se deben mostrar los flujos de entrada y salida del proceso en el nivel 1 (*consistencia con la información de entrada y salida*).



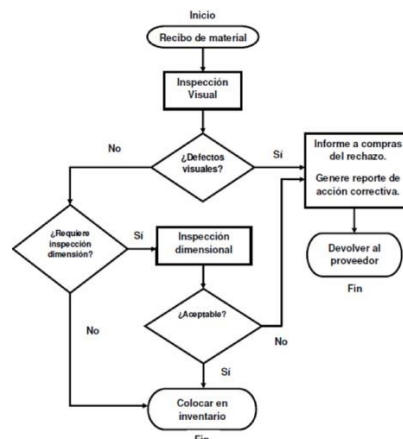
Recomendación:

Se recomienda utilizar un máximo de 4 niveles de descomposición, aunque puede que se necesiten menos.

- Nivel 0: Diagrama de contexto
- Nivel 1: Subsistemas
- Nivel 2: Funciones de cada subsistema
- Nivel 3: Subfunciones asociadas
- Nivel 4: Procesos necesarios para el tratamiento de cada subfunción

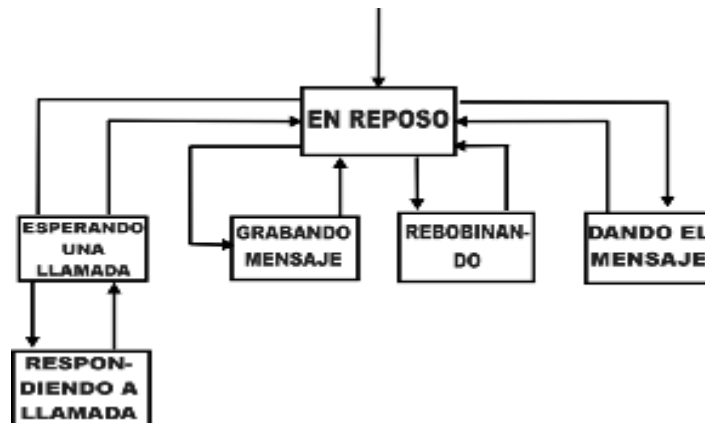
DFC

Similar a los DFD con la diferencia de que muestran el flujo de control, en lugar de datos.



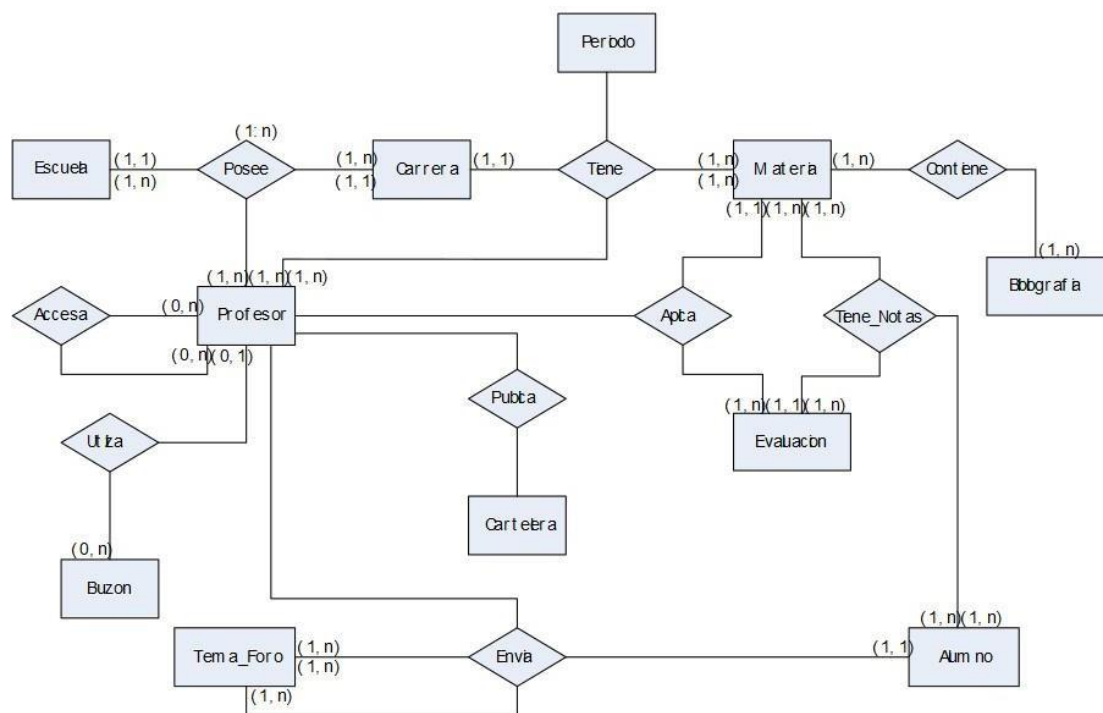
DTE

Representa cómo se comporta el sistema como consecuencia de sucesos externos.



DER

Usado para representar los datos y la forma en la que se relacionan entre ellos.



DD

Es una descripción detallada de los datos utilizados por el sistema que gráficamente se encuentran representados por los flujos de datos y almacenes presentes sobre el conjunto del DFD.

Nombre de Archivo: BDPlantilla

Fecha de creación: 27/04/2008.

Descripción: Base de datos que contendrá la

plantilla de personal del instituto.

Campo	Tamaño	Tipo de Dato	Descripción
CURP	18	Caracter	Clave Unica de Registro de Población
cPaterno	30	Caracter	Apellido paterno del Empleado
cMaterno	30	Caracter	Apellido materno del empleado
cNombre	45	Caracter	Nombre del Empleado.
cDomicilio	60	Caracter	Domicilio actual donde reside el empleado
cColonia	45	Carácter	Colonia del domicilio donde reside el empleado
cCiudad	45	Carácter	Ciudad donde reside el empleado
cEstado	45	Carácter	Entidad federativa de residencia del empleado
cTelefono	12	Carácter	Número telefónico del empleado
nPostal	6	Númerico	Código postal del domicilio del empleado
cFamiliar	65	Carácter	Nombre de un familiar directo del empleado
FDomicilio	65	Carácter	Domicilio de familiar directo del empleado
FTelefono	12	Carácter	Teléfono de familiar directo del empleado

Relaciones:

CURP con BDNomina

Campos Clave:

CURP, cPaterno, cMaterno

Documentación de la fase de Análisis

Todo lo realizado en esta fase debe quedar reflejado en el documento *Especificación de Requisitos del Software (ERS)*.

Debe ser:

- Completo
- Consistente
- Fácil de verificar y modificar
- Fácil de utilizar en otras fases
- Fácil de identificar origen y consecuencias de los requisitos
- Sirve como entrada a la fase de Diseño

Puedes ver la estructura del documento en:

<http://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>

Diseño

Compone la forma en que se solucionará el problema.

Se traducen los requisitos (funcionales y no funcionales) en una representación software.

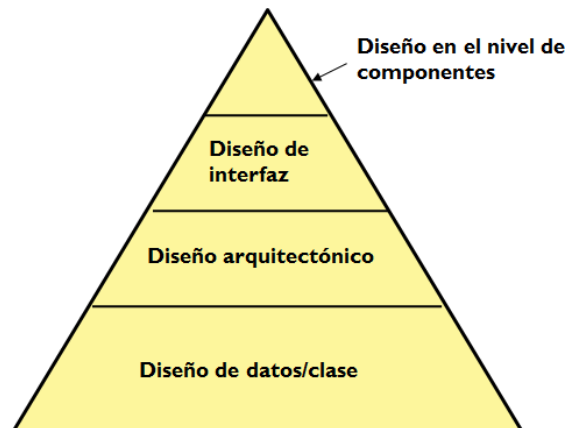
Tipos de diseño:

- **Diseño estructurado:** Basado en el flujo de datos a través del sistema.
- **Diseño orientado a objetos:** Donde el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos.

Diseño estructurado (clásico)

Produce un modelo de diseño de cuatro elementos:

- Diseño de datos/clase
- Diseño arquitectónico
- Diseño de interfaz
- Diseño procedimental / Diseño en el nivel de componentes



Diseño de datos/clase

Se encarga de transformar el modelo de dominio de la información creado durante el análisis, en las estructuras de datos que se utilizarán para implementar el software.

Diseño arquitectónico

Se centra en la representación de la estructura de los componentes software, sus propiedades e interacciones.

Partiendo de los DFD se establece la estructura modular del software que se desarrolla.

Diseño de interfaz

Describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con las personas que lo utilizan.

El resultado de esta tarea es la creación de formatos de pantalla.

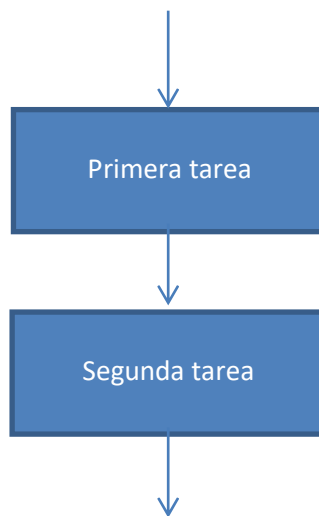
Diseño procedimental/a nivel de componentes

El resultado de esta tarea es el diseño de cada componente software con el suficiente nivel de detalle para que pueda servir de guía en la generación de código fuente en un lenguaje de programación.

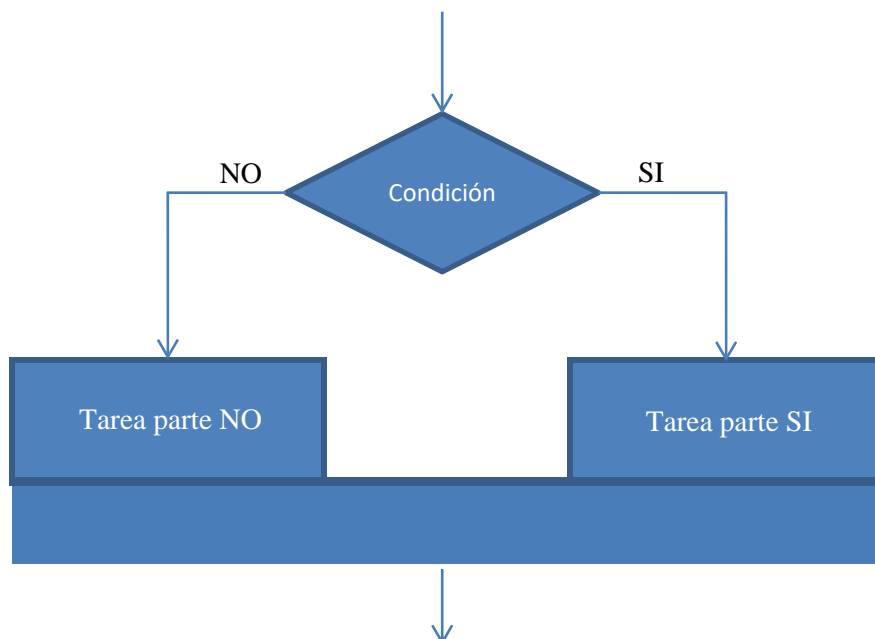
Construcciones lógicas:

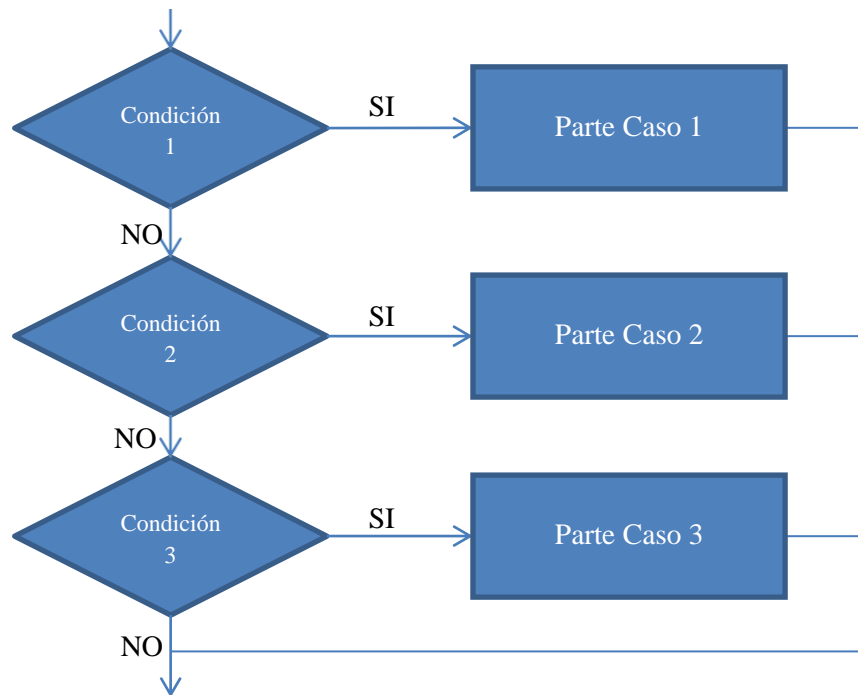
Los fundamentos del diseño a nivel de componentes (procedimental) se establecieron cuando se propuso el uso de un conjunto de construcciones lógicas con las que se podía formar cualquier programa:

1. **Secuencial:** Es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.



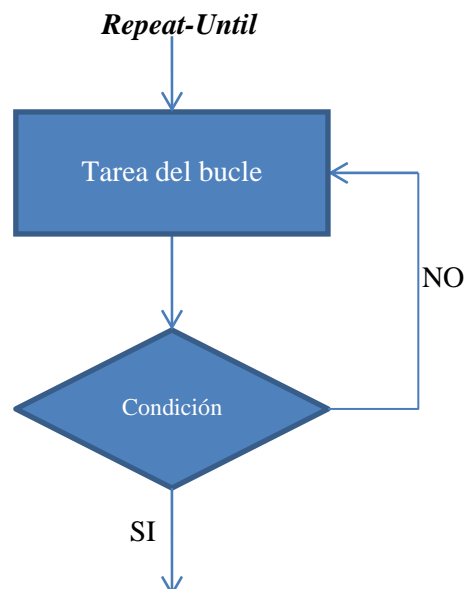
2. **Condicional:** Permite seleccionar un proceso u otro a partir de la evaluación de una condición lógica (rombo).

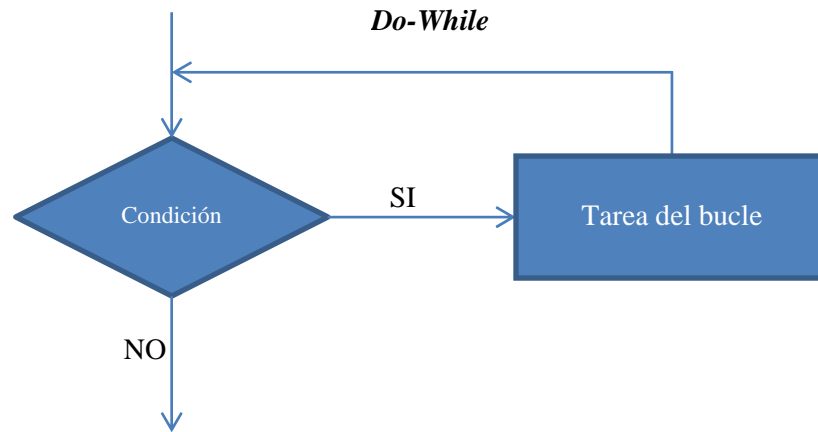




3. **Repetitiva:** Proporciona los bucles:

- Repetir-hasta: ejecuta la tarea y luego comprueba la condición.
- Hacer-mientras: Comprueba la condición y después ejecuta la tarea.
 - Para bucle *for* usamos un *do-while*, donde una tarea del bucle sería incrementar/decrementar el contador.

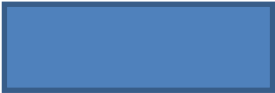















Notaciones gráficas:

Para representar el diseño se emplean algunas herramientas gráficas como son:

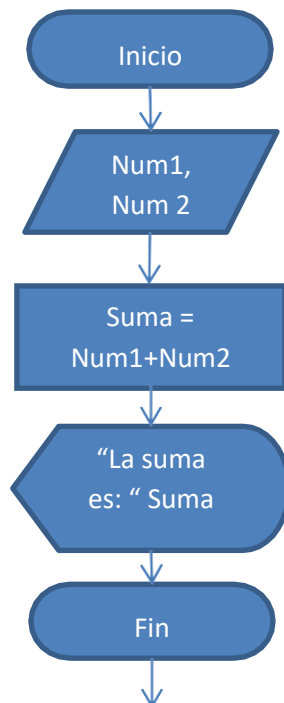
4. **Diagramas de flujo:** Es una herramienta muy usada para el diseño procedimental. Se utilizan los símbolos vistos anteriormente entre otros.

	Proceso
	Condición
	Flujo de control
	Terminador, inicio y final del programa
	Entrada y salida de datos
	Pantalla, representa salida por pantalla
	Impresora, se utiliza como símbolo de entrada de documento y salida
	Teclado, símbolo de entrada por teclado
	Conector, se utiliza para unir una parte del diagrama con otra, dentro de la misma página

	Conector, se utiliza para unir una parte del diagrama con otra, en páginas diferentes
	Llamada a subrutina o procedimiento
	Disco magnético o base de datos, entrada/salida de datos

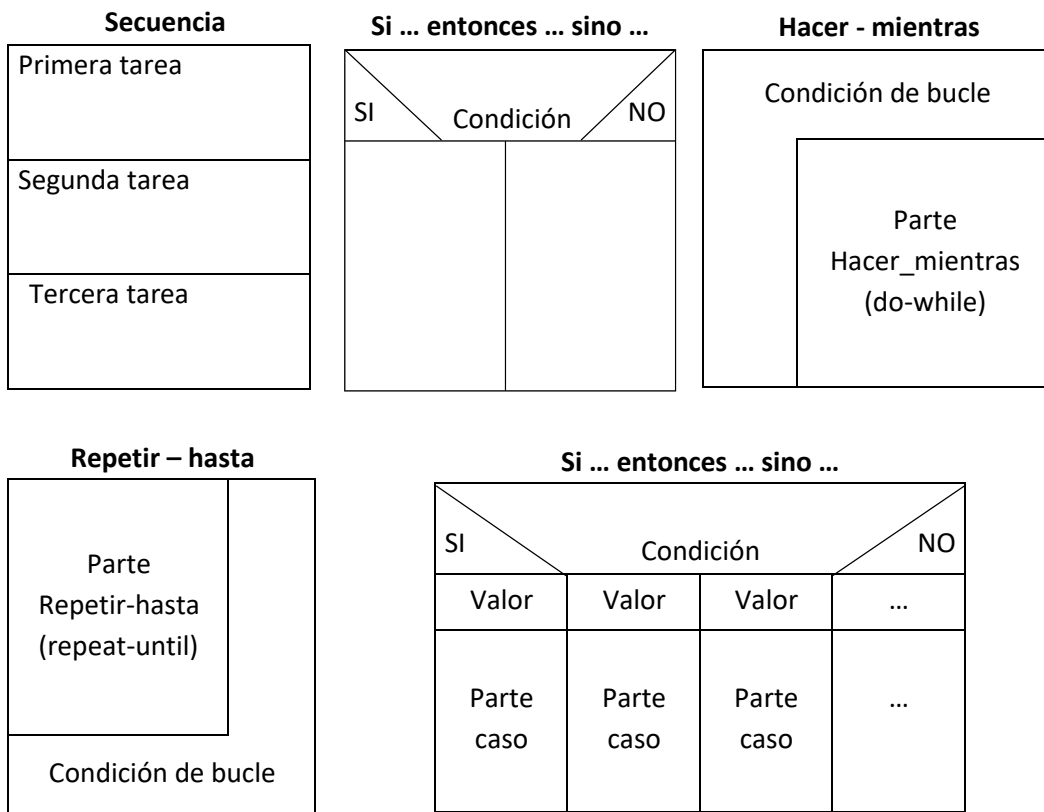
Ejemplo:

Programa que lee dos números y muestra la suma en pantalla. Estructuras secuenciales básicas.



5. Diagramas de cajas:

- Para representar una secuencia se conectan varias cajas seguidas.
- Para una condicional se representa una caja para la parte SI y otra para la parte NO, encima se indica la condición.
- En las repetitivas, el proceso a repetir se encierra en una caja que está dentro de otra caja donde la parte superior (do-while) o inferior (repeat-until) se indica la condición del bucle.
- La selección múltiple, en la parte superior se indica la condición, se definen tantas columnas como valores se vayan a comprobar y debajo de cada valor se indica la parte a realizar.



6. Tablas de decisión:

- Se usan para:
 - Evaluar una combinación compleja de condiciones
 - Seleccionar acciones según estas condiciones
- Se dividen en cuatro cuadrantes:
 - Superior izquierdo: Lista de condiciones
 - Inferior izquierdo: Lista de acciones posibles
 - Superior derecho: Entrada de condiciones
 - Inferior derecho: Entrada de acciones
- Cada columna del cuadrante de la derecha forma una regla de procesamiento

Condiciones	1	2	3	4			N
Condición nº 1	X		X				
Condición nº 2		X	X				
Condición nº 3	X			X			
Acciones							
Acción nº 1	X			X			
Acción nº 2	X	X					
Acción nº 3		X	X	X			
Acción nº 4			X	X			

Ejemplo:

Condiciones	1	2	3	4
Cliente registrado	SI	SI	NO	NO
Importe compra > 800 €	SI	NO	SI	NO
Acciones				
Aplicar 1% bonificación sobre el importe de compra	X	X		
Aplicar 3% descuento sobre el importe de compra	X		X	
Calcular factura	X	X	X	X

Regla 1: Si el cliente SI está registrado y el importe de la compra SI supera 800 €, entonces se aplica un 1% de bonificación y un 3% de descuento sobre el importe de la compra y se calcula la factura.

Regla 2:...

Regla 3:...

Regla 4:...

7. Pseudocódigo:

- Utiliza texto descriptivo para realizar el diseño de un algoritmo
- Mezcla frases del lenguaje natural con estructuras sintácticas que permiten construir las estructuras básicas de la programación estructurada, declarar datos, definir subprogramas,...

Secuencial Instrucción 1 Instrucción 2 ... Instrucción n	Condicional Si <condición> entonces <instrucciones> Si no <instrucciones> Fin si	Condición múltiple Según sea <variable> hacer Caso valor 1: <instrucciones> Caso valor 2: <instrucciones> Caso valor 3: <instrucciones> Otro caso: <instrucciones> Fin según
Repetir – hasta Repetir <instrucciones> Hasta que <condición>	Hacer – mientras Mientras <condición> hacer <instrucciones> Fin mientras	

Ejemplo:

Inicio

Repetir

Visualizar “Escribe dos números”

Leer A, B

Hasta A=B

Fin

Diseño orientado a objetos

Hay que partir de un Análisis Orientado a Objetos (AOO) que define todas las clases, las operaciones, atributos, relaciones, comportamientos y comunicaciones entre clases.

Para el A/DOO se utiliza **UML** (Unified Modeling Language – Lenguaje de Modelado Unificado)

Codificación

1. El programador recibe las especificaciones del diseño
2. Las transforma en un conjunto de instrucciones (lenguaje de programación) → código fuente.
3. Deben existir unas normas de codificación y estilo, claras y homogéneas que faciliten las tareas de corrección y mantenimiento.

Normas (buenas prácticas)

“Cualquier estúpido puede escribir un código que los ordenadores puedan entender. Son los buenos programadores los que escriben códigos que los humanos pueden entender”

(Martin Fowler)

Código limpio: Conjunto de buenas prácticas que pretenden hacer que el código sea más fácil de leer, mantener, extender y menos propenso a errores.

Podemos destacar una serie de normas, aunque existen muchísimas más:

- **Norma 1 – Usa nombres con significado:**
 - El código tiene que poder leerse como si fuera un libro.
 - Nombres de variables, métodos, clases deben dar significado a lo que nuestro programa intenta “contar”.
 - Los nombres de funciones deben ser concretos y claros.
- **Norma 2 – Haz unidades de código pequeñas:**
 - Clases, métodos, funciones,... cortos simplifican la comprensión del código y lo hacen más mantenible.
 - Limitación del número de líneas de métodos, clases, funciones,...
 - Se recomienda no superar más de un nivel de anidamiento.
 - Ej. cada una de las ramas de un if/else debería llamar a una función que, además tenga un nombre con significado.
- **Norma 3 – Las unidades de código deben hacer una única cosa**
 - Principio de Responsabilidad Única (SOLID)
 - Ej. Si una clase hace varias cosas, nos encontraremos con los siguientes problemas:
 - Esta clase será muy propensa a cambios.
 - Nos cuenta más testearla.
 - El código crecerá de forma descontrolada.
- **Norma 4 – Las funciones deben tener un número limitado de argumentos.**
 - Los parámetros añaden una gran cantidad de carga conceptual que dificulta su lectura.

- Dificultan el testing, imagina el número de combinaciones posibles con cuatro parámetros.
 - Preferir funciones sin parámetros.
 - Un máximo de 3 parámetros es un buen límite.
- **Norma 5 - Sigue el principio DRY (Don't Repeat Yourself):**
 - Repetir código es una acción bastante peligrosa que nos causará problemas.
 - Nos obliga a testear lo mismo varias veces.
 - Cuando modificamos algo en un bloque, tendremos que acordarnos de cambiarlo en el resto.
 - Amplían la cantidad de código a mantener.
 - Solución: Extraerlo a una clase o función.
- **Norma 6 – Evita utilizar comentarios siempre que sea posible:**
 - *“Un comentario es un síntoma de no haber conseguido escribir un código claro” (C. Martin)*
 - Antes de escribir un comentario, intenta reescribir el código o nombrar las cosas de otra forma para que el comentario se vuelva irrelevante.
 - OJO! Muchas veces modificamos el código, pero nos olvidamos de hacer lo propio con los comentarios.
 - De todas formas, en muchos casos serán necesarios.
- **Norma 7 – Utiliza un formato único en tu código:**
 - Densidad vertical: los conceptos que están relacionados deberían aparecer juntos verticalmente.
 - Ubicación de los componentes: Ej. normalmente los campos de un clase se suelen poner al principio de la misma.
 - El número de caracteres por línea
 - Un largo etcétera, donde se podrían incluir especificaciones propias de cada lenguaje.

Programas

Un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación que aplicadas sobre un conjunto de datos resuelven un problema o parte del mismo.

Lenguajes de programación

Es un lenguaje formal que proporciona la capacidad de escribir una serie de instrucciones con el fin de controlar el comportamiento físico, lógico y de comunicación de una máquina, de manera que se puedan obtener una serie de resultados.

Consta de:

- **Un alfabeto o léxico:** formado por el conjunto de símbolos permitidos.
- **Una sintaxis:** Reglas que indican cómo realizar las construcciones.
- **Una semántica:** Determina el significado de cualquier construcción.

Clasificación de lenguajes de programación:

Según su nivel de abstracción:

- **Lenguajes de bajo nivel:** Se acercan al funcionamiento de la máquina.
 - **Lenguaje máquina:** Entendible directamente por la máquina. Formados por cadenas de 0s y 1s.
 - **Lenguaje ensamblador:** Necesita ser traducido a lenguaje máquina para poder ejecutarse. Utiliza nombre nemotécnicos y trabaja directamente con registros de memoria física de la máquina.
- **Lenguajes de nivel medio:** Punto intermedio entre bajo y alto nivel. Posee características de ambos. Ej.: C
- **Lenguajes de alto nivel:** Formados por palabras del lenguaje natural (inglés). Para poder ejecutarlos en el ordenador se necesita un programa intérprete o compilador que traduzca las instrucciones escritas en este lenguaje.
 - Son independientes de la máquina, no dependen del hardware del ordenador. Ej. ALGOL, COBOL, C++, Java, PHP, Python,...

Según la forma de ejecución:

- **Lenguajes compilados:** Un compilador es un programa que puede leer un programa escrito en un determinado lenguaje (fuente) y traducirlo en un programa equivalente en otro lenguaje (destino).
- **Lenguajes interpretados:** Los intérpretes dan la apariencia de ejecutar directamente las operaciones especificadas en el programa fuente con las entradas proporcionadas por el usuario. Ej. PHP, JavaScript,...

Según el paradigma de programación:

- **Lenguajes imperativos:** Las instrucciones se ejecutan unas tras otras, de manera secuencial, salvo cuando se encuentran estructuras de control condicionales o bucles. Hay declaración de variables, tipos y procedimientos, aunque esto varía notablemente en función del lenguaje utilizado. Ej. Basic, C++, Java,...
- **Lenguajes funcionales:** Los lenguajes se basan en la declaración de funciones aritméticas. Están compuestos únicamente por la definición de funciones. Ej. Lisp, Miranda, Haskell,...
- **Lenguajes lógicos:** Estudian el uso de la lógica para el planteamiento de problemas y el control sobre las reglas para alcanzar la solución automática. Ej. Prolog.

Prog. Funcional + Prog. Lógica = Prog. Declarativa

- **Lenguajes declarativos:** Basado más en las matemáticas y en la lógica que los lenguajes imperativos. Los lenguajes declarativos no dicen cómo hacer una cosa, sino, más bien, qué cosa hacer. Ej. SQL.
- **Lenguajes estructurados:** Orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa recurriendo únicamente a subrutinas y tres estructuras básicas: secuencias, selección (if y switch) e iteraciones (bucles for y while).

- **Lenguajes orientados a objetos:** Un programa está compuesto por un conjunto de objetos, no por un conjunto de instrucciones como en la programación estructurada.
 - Un objeto consta de una estructura de datos (atributos) y de una colección de métodos que manipulan esos datos.
 - Los objetos se comunican unos con otros a través del paso de mensajes.

Pruebas

En esta etapa ya se dispone del software y, en ella, tratamos de encontrar errores.

Se realizan tareas de verificación y validación del software (V&V).

- **Verificación:** Trata de comprobar si el producto se está construyendo correctamente, es decir, si el software implementa correctamente una función específica.
- **Validación:** Trata de comprobar si el producto es correcto, es decir, si el software se ajusta a los requisitos del cliente.

Casos de prueba y recomendaciones

Un **caso de prueba** es un documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.

Recomendaciones

- Cada prueba debe definir los resultados de salida esperados.
- Un programador debe evitar probar sus propios programas.
- Es necesario revisar los resultados de cada prueba en profundidad.
- Las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.
- Centrar las pruebas en dos objetivos:
 - Comprobar si el software no hace lo que debe hacer
 - Comprobar si el software hace lo que no debe hacer
- Evitar hacer pruebas que no estén documentadas ni diseñadas con cuidado.
- No planear pruebas asumiendo que no se encontrarán errores.
- La probabilidad de encontrar errores es proporcional al número de errores ya encontrados.
- Las pruebas son una tarea creativa.

Diseño de casos de prueba (técnicas)

- **Prueba de caja blanca:** Se centran en validar la estructura interna del programa.
- **Prueba de caja negra:** Se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa.

Proceso de pruebas

1. Generar un plan de pruebas
2. Diseño de pruebas ¿Qué técnicas se emplearán?
3. Generación de los casos de prueba
4. Definición de los procedimientos de la prueba ¿Cómo se va a realizar? ¿Quién...?
5. Ejecución de las pruebas
6. Evaluación
7. Depuración. Localiza y corrige errores.

Documentación

Todas las etapas del desarrollo deben quedar perfectamente documentadas.

Objetivos

- Actuar como un medio de comunicación entre los miembros del equipo de desarrollo.
- Ser un repositorio de información del sistema (mantenimiento).
- Proporcionar información para ayudar a planificar presupuestos, procesos, ...
- Indicar a los usuarios cómo utilizar y administrar el sistema.

Clasificación

Documentación del proceso: Estos documentos registran el proceso de desarrollo y mantenimiento.

Documentación del producto: Estos documentos describen el producto que está siendo desarrollado.

- **Documentación del sistema:** Describe el producto desde un punto de vista técnico.
 - **Fundamentos del sistema:** Describe los objetivos de todo el sistema.
 - **El análisis y especificaciones de requisitos:** Proporciona información exacta sobre la especificación de los requisitos acordados entre las partes interesadas.
 - **Diseño:** Describe la arquitectura del sistema.
 - **Implementación:** Proporciona la descripción de cómo se expresa el sistema en algún lenguaje de programación formal.
 - **Plan de pruebas:** Descripción de cómo se evalúan las unidades de programa individualmente.
 - **Plan de pruebas de aceptación:** Pruebas que debe pasar el sistema antes de que los usuarios lo acepten.
 - **Diccionario de datos:** Descripciones de todos los términos que se relacionan con el sistema.
- **Documentación del usuario:** Describe el producto desde un punto de vista orientado a los usuarios que utilizarán el sistema.
 - **Destinada a:**
 - **Usuarios finales:** Que utilizarán el software.
 - **Usuarios administradores del sistema:** Responsables de la gestión del programa informático.
 - **Documentos:**
 - **Descripción funcional:** Proporciona una descripción general de las funciones del sistema.
 - **Documento de instalación:** Describe cómo instalar el sistema en un entorno particular.
 - **Manual introductorio:** Proporciona explicaciones sencillas de cómo empezar a utilizar el sistema.
 - **Manual de referencia:** Proporciona la descripción detallada de la instalación del sistema y su uso.

- **Guía del administrador:** Debe indicar mensajes generados, comandos, tareas del administrador,...
- **Tarjeta de referencia rápida:** Usado para hacer búsquedas rápidas, ej. Ayuda en línea,...

Estructura del documento

La documentación debe estar correctamente organizada para permitir a los lectores localizar la información más fácilmente.

Organizada en capítulos, secciones, subsecciones,...

Impacto en la legibilidad y usabilidad.

Pautas mínimas generales:

- Portada
- Índice
- Glosario de términos

Estructura IEEE de la documentación de usuario

- **Datos de identificación:** Título, identificado de documento,...
- **Tabla de contenidos**
- **Lista de ilustraciones**
- **Introducción:** Define el propósito del documento y resumen de contenidos.
- **Información para el uso de la documentación**
- **Conceptos de las operaciones:** Explicación conceptual para el uso del software.
- **Procedimientos:** Instrucciones sobre cómo utilizar el software.
- **Mensajes de error y resolución de problemas**
- **Glosario**
- **Fuentes de información relacionada:** Información adicional
- **Características de navegación:** Permiten a los lectores encontrar su ubicación actual y moverse por el documento.
- **Índice:** Lista de términos clave y las páginas en las que estos términos se referencian.
- **Capacidad de búsqueda** (documentación electrónica): Indica la manera de encontrar términos específicos.

Explotación

Una vez que se han realizado las pruebas y documentado todas las etapas pasamos a la fase de Explotación.

Se lleva a cabo la instalación y puesta en marcha del producto en el entorno de trabajo del cliente.

Tareas

1. **Se define la estrategia para la implementación del proceso**
 - Se definen los procedimientos para recibir, registrar, solucionar, hacer un seguimiento de los problemas y para probar el producto en el entorno de trabajo.
2. **Pruebas de operación**
 - Se llevarán a cabo pruebas de funcionamiento y, si se satisfacen, se libera el software para uso operativo.
3. **Uso operacional del sistema**
 - El sistema entrará en acción en el entorno previsto de acuerdo con la documentación de usuario.
4. **Soporte al usuario**
 - Se deberá proporcionar asistencia y consultoría a los usuarios cuando la soliciten.

Mantenimiento

Es la modificación de un producto software después de la entrega para corregir los fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado.

Tipos

- **Adaptativo:** Tiene como objetivo la modificación del producto para adaptar el software a los cambios que se produzcan tanto en el hardware como en el software del entorno en el que se ejecuta.
- **Correctivo:** Tiene como objetivo la corrección de los fallos descubiertos después de la entrega del producto.
- **Perfectivo:** Tiene como objetivo la modificación del producto orientada a incorporar nuevas funcionalidades y nuevas mejoras en el rendimiento o la mantenibilidad del producto.
- **Preventivo:** Tiene como objetivo la modificación del producto sin alterar las especificaciones del mismo, con el fin de mejorar y facilitar las tareas de mantenimiento --> **reingeniería del software**.