



Desarrollo de Aplicaciones con Tecnologías Web

Código: IFCD0210

Familia profesional: Informática y Comunicaciones

Área profesional: Desarrollo

Nivel de cualificación profesional: 3

Relación de módulos formativos y de unidades formativas:

MF0491_3: Programación web en el entorno cliente.(180 horas)

- **UF1841:** Elaboración de documentos web mediante lenguajes de marcas. (60 horas)
- **UF1842:** Desarrollo y reutilización de componentes software y multimedia mediante lenguajes de guión. (90 horas)
- **UF1843:** Aplicación de técnicas de usabilidad y accesibilidad en el entorno cliente (30 horas)

MF0492_3: Programación web en el entorno servidor. (240 horas)

- **UF1844: Desarrollo de aplicaciones web en el entorno servidor. (90 horas)**
- **UF1845:** Acceso a datos en aplicaciones web del entorno servidor. (90 horas)
- **UF1846:** Desarrollo de aplicaciones web distribuidas. (60 horas)

MF0493_3: Implantación de aplicaciones web en entornos internet, intranet y extranet. (90 horas)

MP0391: Módulo de prácticas profesionales no laborales de desarrollo de aplicaciones con tecnología web. (80 horas)

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

1.- El proceso del desarrollo de software

1. *Introducción.*
2. *Modelos del ciclo de vida del software.*
3. *Análisis y especificación de requisitos.*
4. *Diseño.*
5. *Implementación. Conceptos generales de desarrollo de software.*
6. *Validación y verificación de sistemas.*
7. *Pruebas de software.*
8. *Calidad del software.*
9. *Herramientas de uso común para el desarrollo de software.*
10. *Gestión de proyectos de desarrollo de software.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

1.- El proceso del desarrollo de software - Introducción

Ordenador: Hardware + Software

¿Qué tienen en común todos los programas de software? Para su desarrollo se ha debido seguir un **proceso de varias fases en el que, partiendo de unas necesidades, se ha generado un producto operativo que satisface los requisitos iniciales.**

En Ingeniería del Software, un **modelo de proceso de desarrollo de software** puede verse como una **manera de dividir el trabajo en distintas actividades** (o el **ciclo de vida** del producto en distintas fases) **con la intención de lograr la mejor gestión y el mejor resultado para el proyecto.**

A pesar de la variedad de propuestas de proceso de software, existe un **conjunto de actividades fundamentales** que se encuentran presentes en todos ellos:

- **Especificación de software:** Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.
- **Diseño e Implementación:** Se diseña y construye el software de acuerdo a la especificación.
- **Verificación y Validación:** El software debe validarse, para asegurar que cumpla con lo que quiere el cliente y que funciona correctamente.
- **Mantenimiento y Evolución:** Se recoge la corrección de errores posteriores, así como la evolución del programa, para adaptarse a las necesidades del cliente.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.- Modelos del ciclo de vida del software (I).

El término ciclo de vida del software describe el **desarrollo de software, desde la fase inicial hasta la fase final**.

El propósito es **definir las distintas fases** intermedias que se requieren **para validar el desarrollo de la aplicación**, es decir, para **garantizar** que el software **cumpla los requisitos** para la aplicación **y verificación de los procedimientos** de desarrollo: se asegura de que los métodos utilizados son apropiados.

El ciclo de vida **permite que los errores se detecten lo antes posible** y por lo tanto, permite a los **desarrolladores** concentrarse en la **calidad del software**, en los **plazos de implementación** y en los **costos asociados**.

Se debe establecer un **ciclo de vida personalizado para cada proyecto**, teniendo en cuenta las exigencias y características propias del sistema.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

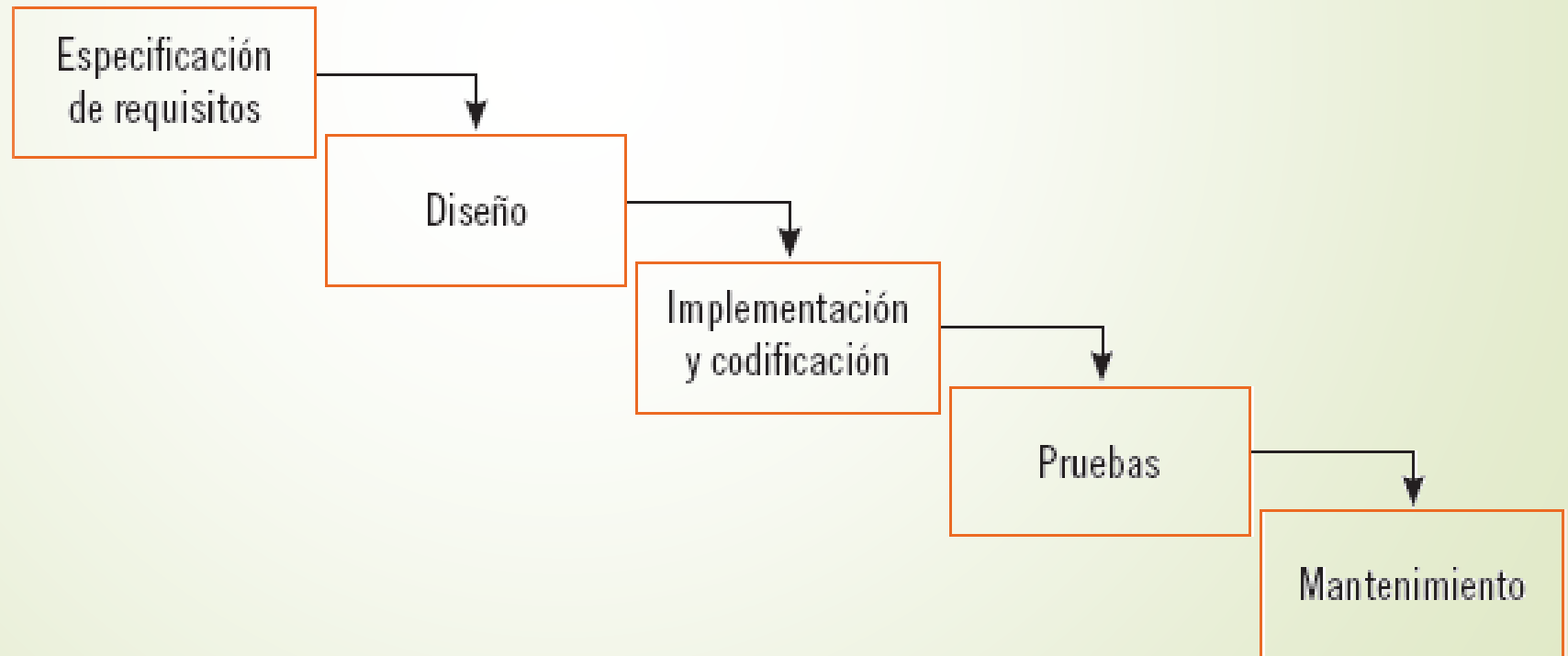
2.1.- Modelos del ciclo de vida del software (II).

Modelos del ciclo de vida:

- *Cascada*
- *Iterativo*
- *Incremental*
- *En V*
- *Basado en componentes (CBSE)*
- *Desarrollo rápido (RAD)*

2.2.- Modelo en Cascada (I)

Este método modela el ciclo convencional de la Ingeniería del Software, aplicando un enfoque sistemático y secuencial de desarrollo que comienza con la ingeniería del sistema y progresa a través del análisis, diseño, codificación, pruebas y mantenimiento.



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.2.- Modelo en Cascada (II)

Las **ventajas** son las siguientes:

- *Planificación sencilla.*
- *Gran calidad del producto final.*
- *Todavía válido para pequeños/medios desarrollos.*

Y los **inconvenientes**:

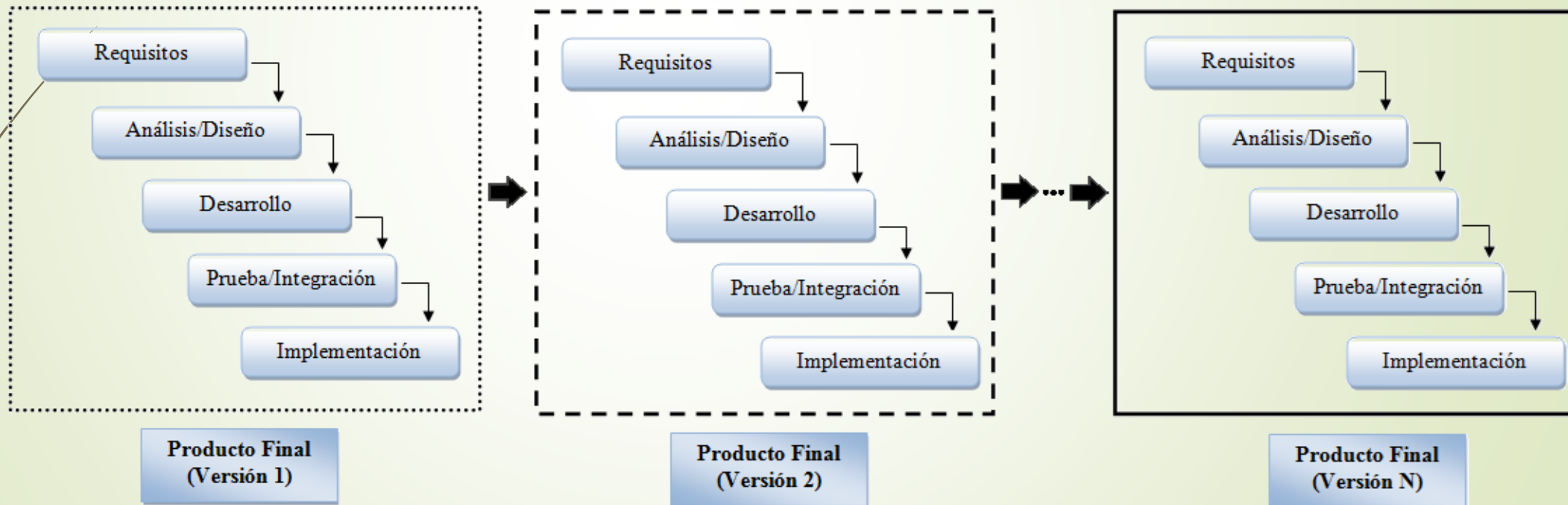
- *Los requerimientos son necesarios al comienzo del proyecto.*
- *Los errores son altamente penalizados.*
- *No se ven resultados hasta una fase avanzada del ciclo.*

Variante: **Modelo en Cascada Retroalimentada**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.3.- Modelo Iterativo (I)

Consiste en una **serie de repeticiones del modelo en cascada**. En cada una de las iteraciones se genera una versión del software, la cual será evaluada y servirá como punto de partida para la siguiente iteración. Cada versión contendrá las mejoras que se consideren, terminando el proceso cuando el producto sea satisfactorio.



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.3.- Modelo Iterativo (II)

Las **ventajas** son las siguientes:

- *Se ofrecen versiones intermedias, evaluables por el cliente.*
- *No requiere una alta especificación de requisitos.*

Y las **desventajas**:

- *Las numerosas versiones pueden encarecer el desarrollo.*
- *Si el cliente se involucra excesivamente puede cambiar su idea, modificando drásticamente en un desarrollo avanzado.*
- *Es difícil establecer un tiempo total de desarrollo.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.4.- Modelo Incremental

Se sigue generando una versión en cada iteración, que servirá de entrada para la siguiente una vez realizada la correspondiente evaluación.

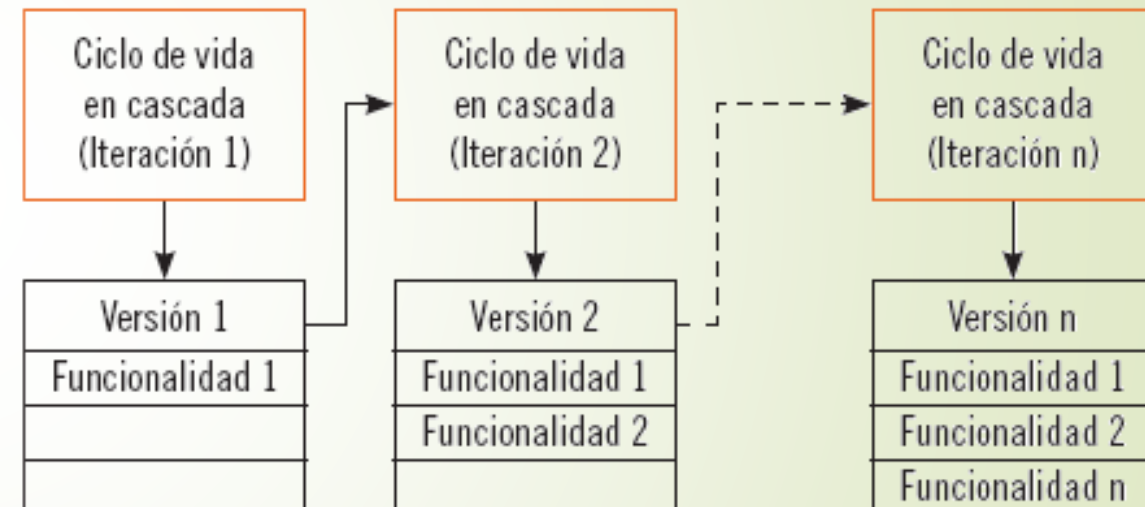
La diferencia radica en que **cada nueva versión conlleva una nueva funcionalidad en forma de módulo.**

Las **ventajas** son las siguientes:

- Entregas rápidas con funcionalidad creciente.

Y las **desventajas**:

- Es difícil establecer el coste y tiempo finales de desarrollo.
- No es válido para cualquier software (software con alta funcionalidad inicial, trabajo en sistemas distribuidos, trabajo en tiempo real, altos requerimientos de seguridad, etc.).



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.5.- Modelo en V

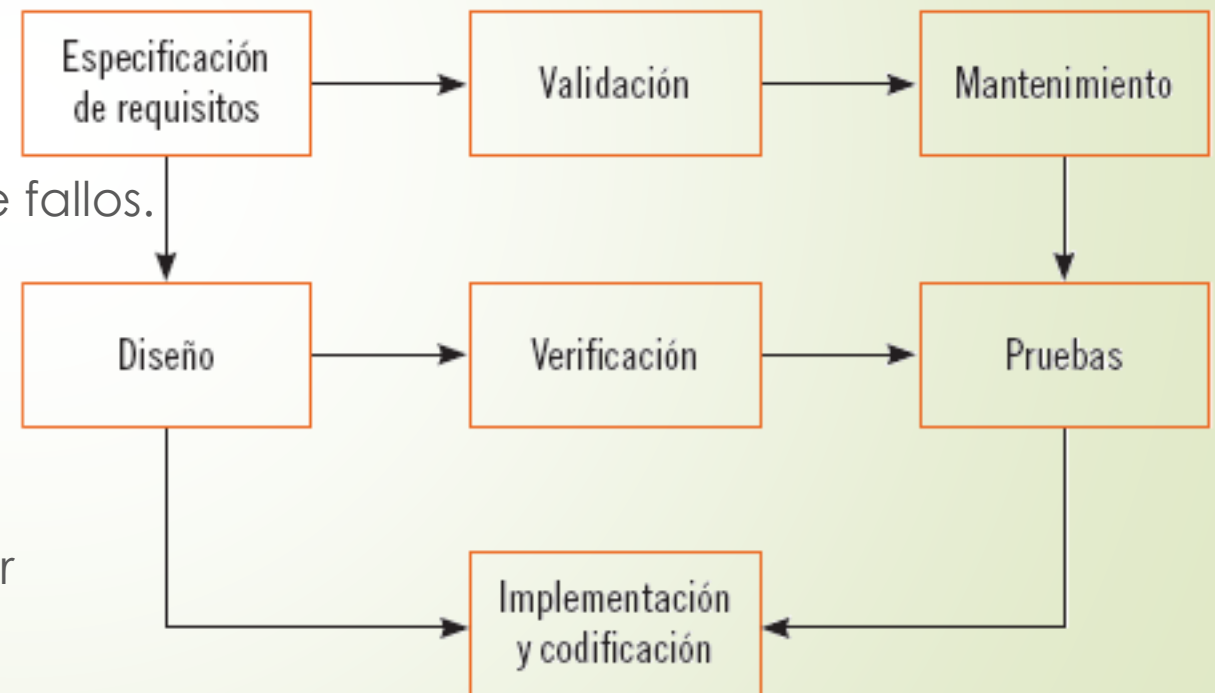
Es igual al modelo en cascada puro, pero incluye dos nuevas etapas entre Especificación de Requisitos y Mantenimiento y entre Diseño y Prueba. Este acercamiento proporciona mayor robustez y garantías, al apoyarse en una constante **verificación y validación**.

Las **ventajas** son las siguientes:

- Las nuevas etapas facilitan la depuración y el control de fallos.

Y las **desventajas**:

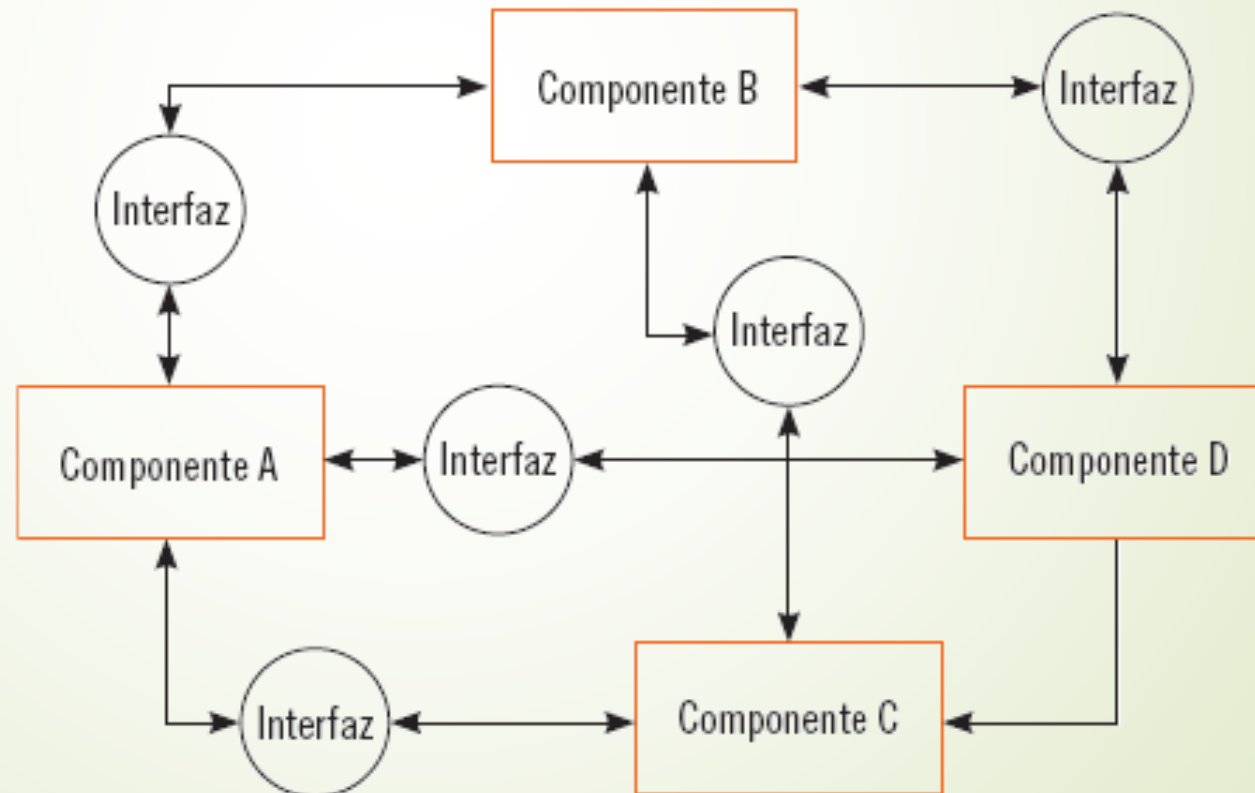
- El cliente no recibirá el producto hasta avanzado el desarrollo.
- Las pruebas pueden disparar el coste de desarrollo.



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.5.- Basado en componentes (CBSE) (I)

Def. (Szypersky-1998): “Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio”



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.5.- Basado en componentes (CBSE) (II)

Las **ventajas** son las siguientes:

- Reutilización de software.
- Se puede reducir el tiempo de desarrollo.
- Simplifica las pruebas y mejora la calidad.

Y los **inconvenientes**:

- El desarrollo depende de las limitaciones de los componentes.
- Los componentes pueden ser caros.
- El componente, al ser externo, puede no ser actualizado durante la vida del software.
- Hay que realizar pruebas exhaustivas

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.6.- Desarrollo rápido (RAD-Rapid Application Development) (I)

Un ciclo de desarrollo rápido prácticamente **fusiona las fases de análisis y diseño**, requiriendo la **colaboración absoluta del cliente**. Este recibirá **muchos prototipos** o partes funcionales de su software **en cortos espacios de tiempo**, evaluando el producto para dar lugar a una retroalimentación en el equipo de desarrollo. De ahí que surja el **concepto** de **adaptación incremental**, puesto que **se espera que los requisitos cambien** (y lo harán, sin duda).

Las **ventajas** son las siguientes:

- *Resultados rápidamente visibles.*
- *Permite la reusabilidad de código.*
- *Participación del usuario.*

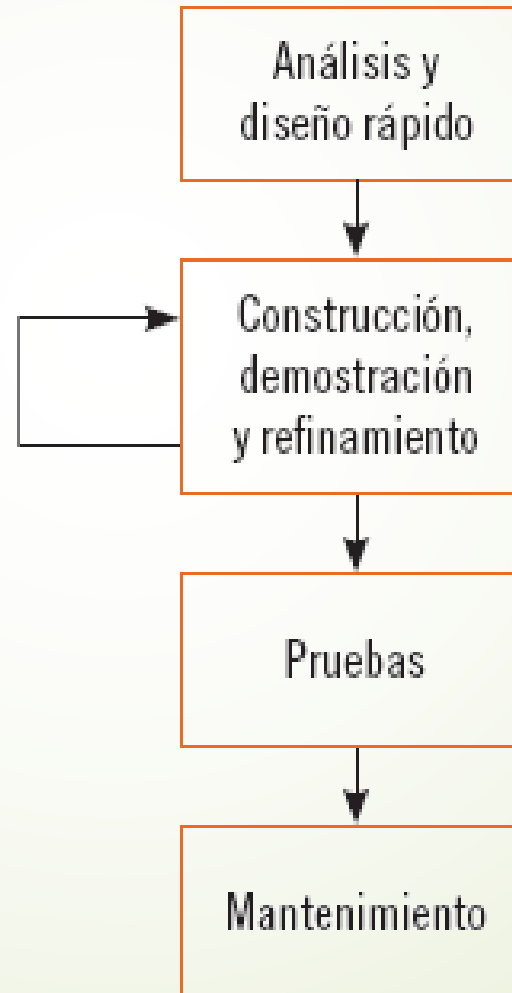
Y los **inconvenientes**:

- *Exige bastante disciplina y compromiso por todas las partes.*
- *Gran coste en herramientas y entornos de desarrollo.*
- *Si el proyecto es grande, son necesarios varios equipos de trabajo.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.6.- Desarrollo rápido (RAD-Rapid Application Development) (II)

60 – 90 días



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.7.- Ventajas e inconvenientes. Pautas para la selección de la metodología más adecuada (I)

En la elección del ciclo de vida adecuado, hay que considerar **cinco factores** básicos:

- **Tiempo** hasta la entrega final.
- **Complejidad** del problema.
- Necesidad (o no) de **entregas** parciales.
- Definición y exactitud de los **requerimientos**.
- **Recursos** disponibles.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

2.7.- Ventajas e inconvenientes. Pautas para la selección de la metodología más adecuada (II)

La valoración conjunta de estos cinco factores permite establecer las siguientes preferencias:

- **Modelo en cascada:** desarrollos no excesivamente complejos, sin entregas parciales y requisitos claramente definidos.
- **Modelo iterativo:** desarrollos para los cuales no se tienen suficientemente claros los requisitos o bien estos son cambiantes. Este modelo favorece la creación de prototipos e involucra al cliente desde el principio.
- **Modelo incremental:** recomendable para software que no requiera toda su funcionalidad de manera inicial y cuando se precisen entregas rápidas.
- **Modelo en V:** desarrollos que requieran gran robustez y confiabilidad. Por ejemplo, software que precise de operaciones constantes sobre una base de datos.
- **Modelo basado en componentes (CBSE):** aplicable a desarrollos que han contemplado en su diseño la incorporación de componentes comerciales, siempre que se pueda afrontar su coste.
- **Desarrollo rápido de aplicaciones (RAD):** desarrollos rápidos, siempre que se disponga de suficiente equipo para hacer frente y cumplir los plazos.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.- *Análisis y especificación de requisitos (I).*

Según el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos), **un requisito** encaja en las siguientes definiciones:

- *Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.*
- *Una condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta.*
- *Una representación documentada de una condición o capacidad.*

Al conjunto de técnicas y actividades que tienen como objetivo descubrir y entender los requisitos de un proyecto se llama **Ingeniería de Requisitos**.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.1.- Tipos de requisitos:

- **Requisitos de usuario:** descripción en lenguaje natural (o mediante diagramas simples) de servicios y funcionalidades que se esperan del software. Por ejemplo: el sistema puede guardar imágenes en archivos.
- **Requisitos de sistema:** especificación completa de los anteriores, que sirve como contrato entre el cliente y el desarrollador. Tomando el ejemplo anterior, se puede plantear:
 - El usuario elige la imagen a guardar.
 - El usuario puede elegir el formato de imagen.
 - El usuario puede indicar el nombre del nuevo archivo.
 - El usuario debe confirmar los datos anteriores para el guardado definitivo de la imagen.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.1.- Tipos de requisitos:

Los requisitos del sistema, a su vez, se pueden dividir de la siguiente manera según la naturaleza del requisito:

- **Requerimientos funcionales:** descripción de la funcionalidad, del comportamiento del sistema y de su interacción con el entorno. En la medida de lo posible, hay que ceñirse a lo que el sistema debe hacer (o qué no debe hacer). El cómo queda para fases posteriores. Ejemplos:
 - *El acceso al sistema requiere darse de alta.*
 - *Se permite un máximo de dos libros simultáneos en préstamo por cada usuario.*
 - *El usuario puede consultar su historial de préstamos.*
 - *Un retraso en la devolución de un libro de más de una semana implica la inhabilitación automática de la cuenta de usuario.*
- **Requisitos NO funcionales:** restricciones que afectan al sistema (estándares, rendimiento, accesibilidad, interfaz, seguridad, portabilidad, etc.). Ejemplos:
 - *Compatible con Internet Explorer 6.0.*
 - *La base de datos debe estar implementada con MySQL.*
 - *Debe cumplir con lo dispuesto en la Ley Orgánica de Protección de Datos.*
 - *El sistema será accesible a través de smartphones y dispositivos móviles, con un interfaz especialmente diseñado para ello.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.2.- Modelos para el análisis de requisitos (I)

Una vez los requisitos han sido definidos, se procede a un modelado de los mismos con dos objetivos: **delimitar el alcance del sistema y capturar su funcionalidad**. Existen muchos modelos que ayudan en el análisis de requisitos. Entre los más importantes se encuentran:

- **Diagramas de flujo:** representa el flujo de la información desde que entra en un sistema hasta que sale, indicando las transformaciones (burbujas) que sufre dicha información al moverse dentro del sistema.
- **Casos de uso:** según el nivel de detalle que se busque:
 - **Diagramas de casos de uso:** muestran la relación entre los casos de uso y los actores, representando una funcionalidad básica del sistema.
 - **Especificación de casos de uso:** descripción en lenguaje fácilmente comprensible, proporcionando información más detallada.

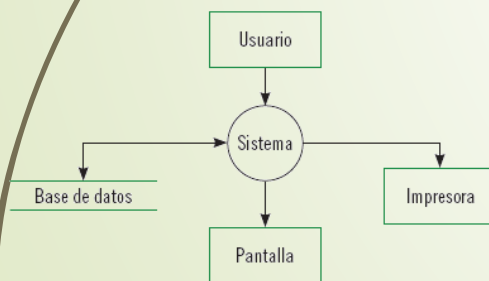
3.2.- Modelos para el análisis de requisitos (II)

Diagramas de flujo: Consta de cuatro componentes básicos:

- **Procesos:** componente que transforma a la información (círculo).
- **Flujo de datos:** indica comunicación entre componentes (flecha).
- **Almacenes de datos:** información del sistema (dos líneas horizontales paralelas).
- **Entidades externas:** receptores o generadores de información (cuadrado).

Un diagrama de flujo tiene **varios niveles, según el grado de detalle** que haga gala:

Diagrama de Flujo de Nivel 0



ejemplo

1. **Nivel 0 (diagrama de contexto):** se representa el sistema como un único proceso y las interacciones con el resto de entidades (usando una flecha indicando el sentido de la interacción).
2. **Nivel 1 (diagrama de nivel superior):** se indican los procesos que describen al proceso principal, siendo este desglosado en subprocesos.
3. **Nivel 2 (diagrama de detalle o expansión):** se aumenta el nivel de detalle, indicando excepciones y flujos entre procesos.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.2.- Modelos para el análisis de requisitos (III)

Casos de uso: **Diagramas de casos de uso:** relación Caso de uso – Actor/es

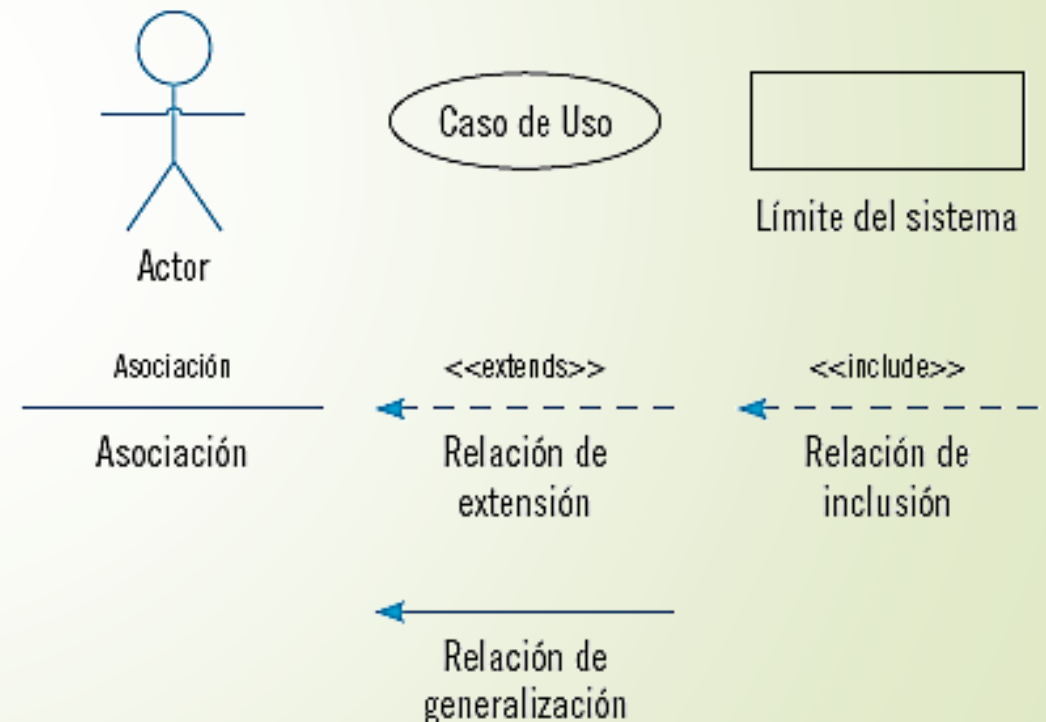
Actor: Entidad externa o interna que interactúa con el sistema, ya sea haciendo uso de una funcionalidad (usuario) o actuando de supervisor o apoyo (base de datos externa).

Proceso **realización correcta:**

1. Identificar **actor/es principal/es**.
2. Identificar el **objetivo** de cada actor respecto al sistema.
3. Presentar/describir **como interactuará** el actor con el sistema.

ejemplo

Componentes de un diagrama de casos de uso



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.2.- Modelos para el análisis de requisitos (IV)

Casos de uso: **Especificación de casos de uso:** *descripción comprensible aportando información detallada.*

Una correcta especificación contendrá:

- Nombre del caso de uso e Identificador.
- Actores.
- Tipo (primario, secundario u opcional).
- Referencias: requisitos que se pueden incluir.
- Precondición: condiciones para que tenga lugar el caso.
- Postcondición: efectos inmediatos sobre el sistema.
- Autor, fecha y versión.
- Propósito: descripción general del caso de uso.
- Flujo normal: curso normal del caso de uso.
- Flujos alternos: cursos alternativos (excepciones).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.3.- Documentación de requisitos.

Los requisitos se deben recoger en el documento ERS (Especificación de Requisitos Software), sin mencionar ningún detalle de diseño. Según el estándar IEEE 830, un documento de ERS debe tener las siguientes características:

- Utilizar términos únicos (en caso de que no sea posible indicarlo en un glosario).
- Incluir todos los requisitos significativos.
- Requisitos fáciles de verificar por un procedimiento existente.
- Los requisitos no deben entrar en conflicto.
- Requisitos clasificados por orden de importancia.
- Fácil de modificar.
- Facilidad de referencia con otros elementos del ciclo de vida.
- Facilidad de utilización en Explotación y Mantenimiento.

Dicho estándar establece también la estructura recomendada para el documento de ERS, que se puede sintetizar en los siguientes puntos:

1. Introducción.
2. Descripción general.
3. Requisitos específicos.
4. Apéndices.
5. Índice.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.4.- Validación de requisitos.

Los requisitos deben ser validados para asegurar que el documento de ERS define el software de manera adecuada. Las principales técnicas de validación son:

- **Revisiones:** revisión de la documentación o del modelado de requisitos **por parte de un grupo de personas** (con participación de representantes del cliente y de usuarios). Se puede utilizar una lista de comprobación definida al comienzo del proceso con el fin de acotar el ámbito de la revisión.
- **Prototipos:** en ocasiones, durante la definición de requisitos, se opta por desarrollar un prototipo del software que **no dispone de la funcionalidad completa**. Así, el usuario/cliente puede tener una perspectiva del sistema.
- **Pruebas de aceptación:** pruebas **previas al paso de producción**. Se recomienda que para estas pruebas no formen parte las personas encargadas del desarrollo, con el fin de que sean lo más objetivas posibles y no haya conflicto de intereses.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

3.5.- Gestión de requisitos.

El uso de **herramientas CASE** es muy útil durante este proceso, ya que facilita la generación de documentación y el seguimiento de las modificaciones.

*Las herramientas **CASE** (Computer Aided Software Engineering) son aplicaciones que ayudan durante el proceso de desarrollo del software, reduciendo el coste de dicho desarrollo y aumentando la productividad.*

El proceso de gestión de cambio consta de los siguientes puntos:

1. Comprobar la validez del cambio.
2. Crear una lista de requisitos afectados por la modificación.
3. Proponer cambios de requisitos de la lista anterior.
4. Valorar los costes de los cambios.
5. Valorar la aceptabilidad de los cambios.
6. Implementar los cambios.

Los puntos 3 y 5 requieren la participación del cliente. Este debe dar el visto bueno a los cambios propuestos y, posteriormente, aceptar el coste de dicha modificación.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.- Diseño.

Una vez definidos los requisitos del sistema, se procede a realizar un modelado del sistema. Aquí **se definirán**, entre otros aspectos, los **componentes que darán cuerpo al sistema, la relación entre los mismos y el interfaz de usuario**.

4.1. Modelos para el diseño de sistemas

Según Pressman, **el objetivo del diseño es realizar un modelo de la entidad que se va a construir**. Para llevar a cabo este objetivo, divide el modelo de diseño en otros tres modelos:

- modelo de arquitectura
- modelo de componentes
- modelo de interfaz.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.1. Modelos para el diseño de sistemas (I)

Modelo de arquitectura: *define la estructura de los componentes del sistema, sus propiedades externas visibles y las relaciones entre ellos.*

- **Centrada en datos:** existe un almacén central de datos al cual acceden el resto de componentes del sistema.
- **De flujo de datos:** los datos se transforman a través de los componentes. Si solo hay un flujo se llama lote secuencial.
- **De programa principal/subprograma:** un programa principal invoca a otros componentes y estos, a su vez, invocan a otros componentes diferentes. La arquitectura, si presenta una distribución remota, recibe la denominación de llamada de procedimiento remoto.
- **Orientada a objetos:** un programa principal se apoya en objetos para llevar a cabo su propósito. Los objetos encapsulan su funcionalidad, comunicándose a través de una interfaz bien definida.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.1. Modelos para el diseño de sistemas (II)

Modelo de componentes: Un componente es definido como una *parte modular, desplegable y sustituible de un sistema, que incluye la implantación y expone un conjunto de interfaces*.

El componente tiene matices en su concepción, dependiendo de la orientación dada al software:

- Visión orientada a objetos.
- Visión tradicional.

ej.

Visión orientada a objetos: Los componentes implementan el funcionamiento de una entidad llamada **Clase**, la cual consta de unos **atributos** (para almacenar información) y de unos **métodos** (que proporcionan funcionalidad). También existen unos métodos especiales que les permiten relacionarse con el exterior (**interfaces**).

ej.

Visión tradicional: El **componente** es un elemento funcional que **incorpora la lógica y las interfaces** de entrada/salida de datos. La **diferencia** respecto a la visión anterior es que *aquí se implementa una funcionalidad, mientras que un componente orientado a objetos puede implementar todas las funcionalidades definidas en su clase*.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.1. Modelos para el diseño de sistemas (III)

Modelo de interfaz: La interfaz **define la forma en la que el usuario interactuará con el software**, tanto proporcionando información como recibiendo resultados

Theo Mandel definió las **Tres reglas doradas** del diseño de la interfaz:

1. *Dejar el control al usuario.*
2. *Reducir la carga de memoria del usuario.*
3. *Hacer que la interfaz sea consistente.*

Para **diseñar una interfaz**, se puede partir de los casos de uso. El proceso de diseño implica:

1. *Identificar al usuario que interactúa con el sistema.*
2. *Identificar sus acciones y los objetos que intervienen en dichas acciones.*
3. *Convertir los objetos en opciones visuales, definiendo, a su vez, las acciones dentro de dichas opciones.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.2. Diagramas de diseño. El estándar UML

El UML (Lenguaje de Modelado Unificado) se define como un ***lenguaje estándar para escribir diseños de software***, creado para ayudar a diseñadores y desarrolladores en el manejo de un lenguaje común. La versión 2.0 proporciona trece diagramas, se analizarán los más utilizados:

1. **Diagramas de casos de uso:** ayudan a determinar la funcionalidad del sistema desde la perspectiva del usuario (ya vistos).
- ej. 2. **Diagramas de implementación:** Muestran la distribución física de un sistema de software cuando intervienen diferentes componentes hardware.
- ej. 3. **Diagramas de actividad:** muestra el comportamiento dinámico de una actividad que se desea modelar
- ej. 4. **Diagramas de estado:** modelan los estados de un objeto, indicando cuáles son las acciones que provocan las transiciones entre los mismos y cuál es el evento disparador que detona el estado inicial.
- ej. 5. **Diagramas de secuencia:** muestran la comunicación de los objetos durante la ejecución de una tarea.
- ej. 6. **Diagramas de comunicación:** similar a los diagramas de secuencia, pero enfatizando la relación entre objetos en lugar del orden temporal.



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

4.3. Documentación

El **Documento de Diseño de Software** (SDD) muestra cómo el sistema estará estructurado para cumplir los requisitos definidos en la fase anterior. Es la principal referencia del programador para empezar a escribir código. Un SDD deberá contener la siguiente información:

- 1. Introducción:** Propósito, Ámbito, Descripción general del documento.
- 2. Descripción general del sistema:** funcionalidad, contexto y diseño del sistema.
- 3. Arquitectura del sistema:** Diseño arquitectónico, Diseño arquitectónico de los subsistemas, Criterio de diseño.
- 4. Diseño de datos:** Descripción de datos, Diccionario de datos
- 5. Diseño de componentes:** diseño de cada componente usado en el sistema.
- 6. Diseño de interfaz:** descripción detallada de la interfaz, mostrando capturas e indicando cuáles son los objetos sobre los que se puede interactuar. Mostrará además la funcionalidad asociada (todo ello desde la perspectiva del usuario).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.- Implementación. Conceptos generales de desarrollo de software.

Una vez que se dispone de la arquitectura del sistema, con la especificación de todas las entidades a desarrollar, entra en juego la codificación. Aquí la idea es ***hacer una implementación, mediante un lenguaje de programación, para generar algo ejecutable.***

Sabía que... La creencia popular es que la fase de codificación ocupa la mayor parte del proceso de desarrollo. Si se parte de buenos diseños previos, esta fase puede ser muy corta.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.1.- Técnicas de desarrollo de software.

Las técnicas de desarrollo de software se definen como **el medio escogido para codificar una serie de órdenes lógicas en un determinado lenguaje de programación**. Se establecen los siguientes tipos:

- **Programación imperativa:** el programa detalla los pasos a seguir, en forma de conjunto de instrucciones, para llevar a cabo una tarea. Es decir, se le dice al programa qué hacer.
 - Programación Estructurada
 - Programación Orientada a Objetos
- **Programación declarativa:** el programa describe los mecanismos a utilizar y el resultado a obtener, pero no implementa los pasos a seguir. Es el tipo de programación utilizada en lenguajes de Inteligencia Artificial y en lenguajes de consulta sobre base de datos.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.2.- Principios básicos del desarrollo de software.

los **Siete principios básicos del desarrollo software**, que fueron desarrollados por Barry Boehm con el fin de concretar en ellos una larga lista de aforismos y buenas costumbres:

1. *Usar un plan de desarrollo basado en un ciclo de vida*
2. *Realizar una continua validación*
3. *Mantener un control del producto*
4. *Usar técnicas de programación modernas*
5. *Mantener un control de resultados*
6. *Mucha gente no garantiza un mejor proyecto*
7. *Comprometerse a mejorar el proceso de desarrollo*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (I).

En este apartado, se indicarán los principios básicos de la programación estructurada. Como se dijo anteriormente, muchas de estas ideas son compartidas por la programación orientada a objetos.

Inicialmente, se mostrarán las **estructuras de control** sobre las que se apoya el Teorema del programa estructurado de Böhm y Jacopini. Después se hará un breve recorrido por los **tipos de datos**, para seguir por los **operadores** y terminar con las **funciones y procedimientos**.

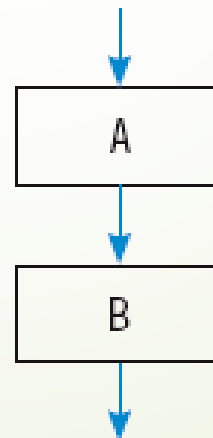
5.3.- Principios básicos de programación (II).

Estructuras de control

Una estructura incide directamente sobre el flujo de ejecución, modificándolo en función de ciertas condiciones. Con la combinación de estas estructuras, se puede desarrollar cualquier programa.

Secuencia: Es, simplemente, la ejecución secuencial de una serie de instrucciones.

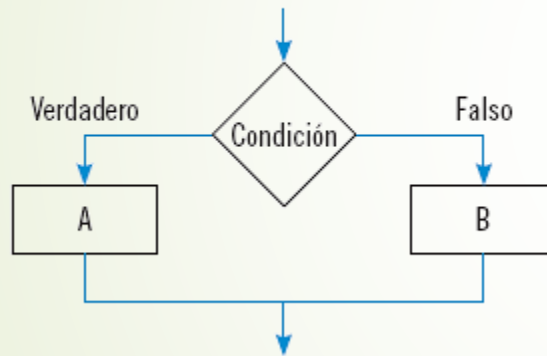
Estructura secuencial



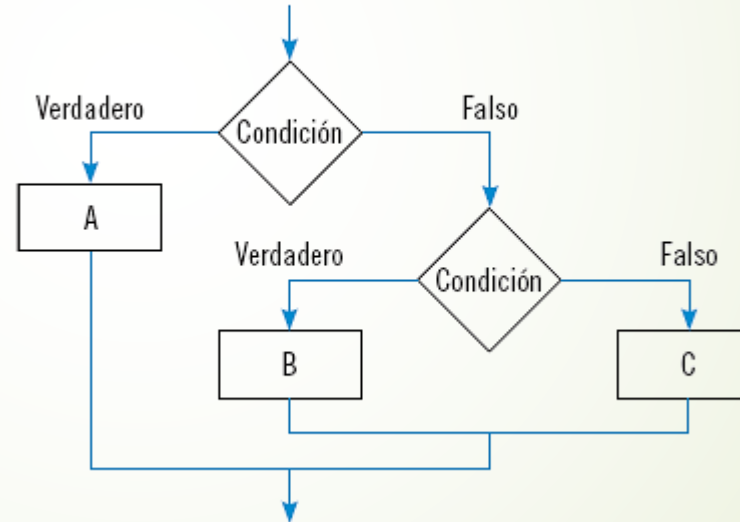
5.3.- Principios básicos de programación (III).

Selección: Se evalúa una condición para determinar por dónde continuará el flujo del programa.

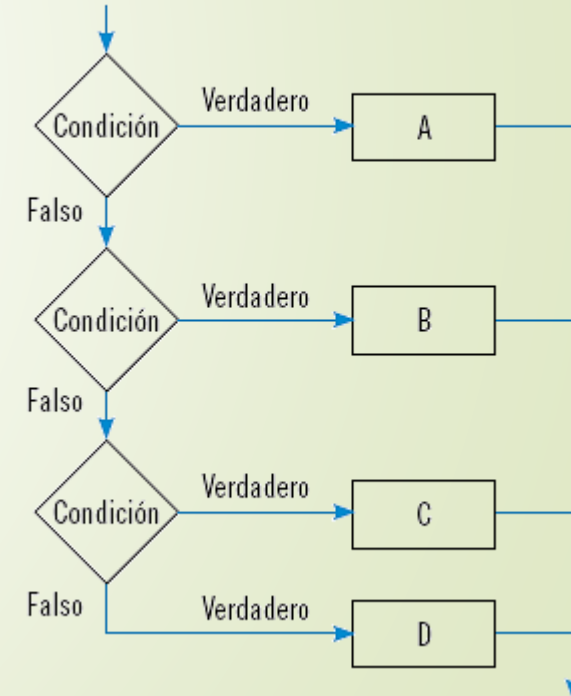
Estructura IF-ELSE #1



Estructura IF-ELSE #2



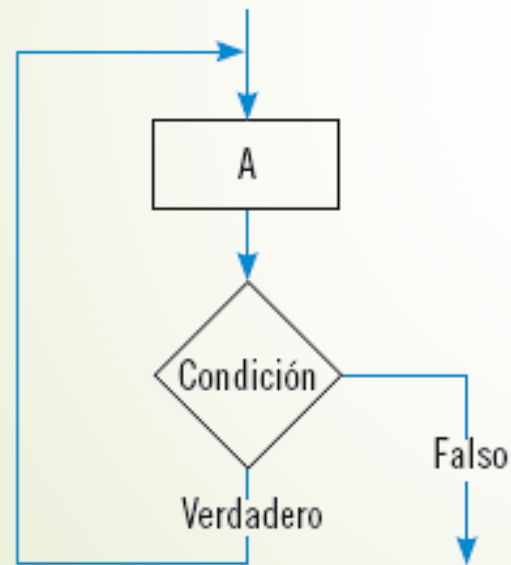
Estructura SWITCH



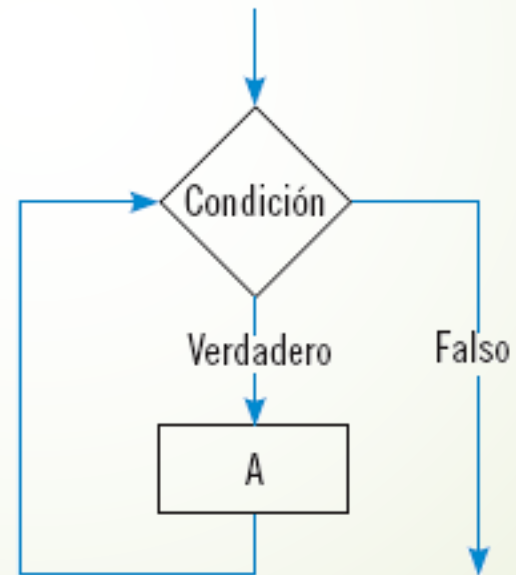
5.3.- Principios básicos de programación (IV).

Iteración: En este bloque de control, se repite la ejecución de una instrucción mientras se cumpla cierta condición. .

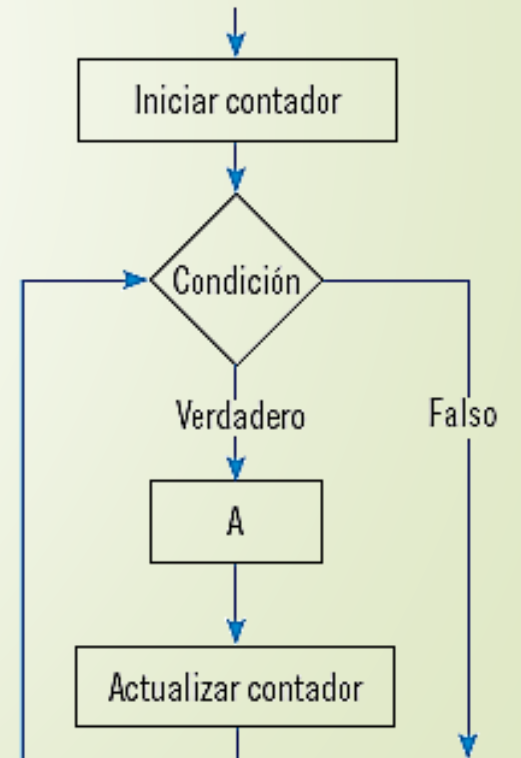
Estructura DO-WHILE



Estructura WHILE



Estructura FOR



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (V).

Tipo de datos

Un tipo de datos es un **atributo que indica el tipo de dato con el cual se va a trabajar**. La distinción clásica establece dos tipos de datos:

- **Primitivos:** tipos de datos básicos que proporciona el lenguaje de programación y que pueden ser usados para crear tipos de datos compuestos.
 - **Carácter (char)**
 - **Entero (int)**
 - **Real (float)**
 - **Booleano (boolean)**
- **Compuestos:** datos generados a partir de los tipos primitivos.
 - **Cadena de caracteres (o string)**
 - **Vector (o array unidimensional)**
 - **Matriz (o array bidimensional)**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (VI).

Variable: Una variable es un contenedor de datos que almacena información de un tipo de dato determinado. Para poder trabajar con los datos hay que utilizar variables.

Cuando se dispone de una variable se le puede asignar un valor. Este valor se almacenará en la posición de memoria referencia por la variable. Existen dos **posibilidades de asignación**:

- *Definir la variable e inicializar al mismo tiempo*

```
int contador = 1;
```

- *Asignar un valor después de la definición*

```
int contador;
```

```
contador = 1;
```

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (VII).

Operadores

Los operadores son **elementos que permiten modificar el valor de las variables**. Algunos operan con una única variable, mientras que otros necesitan de dos.

Existen cuatro tipos:

- **Operadores matemáticos.**
- **Operadores relacionales.**
- **Operadores lógicos.**
- **Operador de asignación.**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (VIII).

- **Operadores matemáticos.**

Permiten realizar operaciones matemáticas con los valores contenidos en las variables. Se pueden utilizar siempre que estas contengan un valor numérico.

Los más usados son los siguientes:

- **Suma** (+).
- **Resta** (-).
- **Multiplicación** (*).
- **División** (/).
- **Módulo** (mod). El módulo es el resto de la división entera.
- **Exponenciación** (^).

Reglas de prioridad, debe seguir el siguiente orden:

1. **Tiene preferencia lo que haya dentro de un paréntesis.**
2. **Exponenciación.**
3. **Multiplicación, división y módulo.**
4. **Suma y resta.**

Ej: $4 + 4 * 5 = 24$ pero con paréntesis $(4 + 4) * 5 = 40$

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (IX).

- **Operadores relacionales.**

Establecen una **relación entre los valores contenidos en dos variables**. De esta relación **se obtiene un resultado (verdadero o falso)**. Las dos variables tienen que ser del mismo tipo. En el caso de no especificar con paréntesis, los operadores relacionales tienen menor prioridad que los aritméticos:

- **Menor que (<)**
- **Mayor que (>)**
- **Menor o igual que (<=)**
- **Mayor o igual que (>=)**
- **Distinto (!=)**
- **Igual (==)**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (X).

- **Operadores lógicos**

Sirven para combinar condiciones. Indispensables para usar en las estructuras de control vistas anteriormente:

- **Operador AND (&&):** las dos condiciones deben ser ciertas.
- **Operador OR (| |):** basta con que sea cierta una condición.
- **Operador NOT (!):** cierta si la condición no se cumple.

- **Operador de asignación.**

El operador básico de asignación es "**=**". El operando de la izquierda se establece con el valor de la expresión de la derecha.

curiosidad: `a = (b = 4) + 5;` // ahora **a** es igual a 9 y **b** se ha establecido en 4

Dependiendo del lenguaje, podemos encontrar otros operadores, se verán más adelante (`+=` `-=` `*=` `/=` `%=`).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

5.3.- Principios básicos de programación (XI).

Procedimientos y Funciones

Una función es una **parte de código que realiza una tarea**. Puede recibir (o no) **parámetro/s**, pero **siempre debe devolver algo**.

Un procedimiento es lo **mismo que** una **función**, con la salvedad de que **no tiene que devolver un valor**.

Esta distinción es la separación histórica entre función y procedimiento, pero existen matices según el lenguaje de programación.

Por **ejemplo**, en **C++**, solo existe el concepto de función, puesto que siempre hay que devolver algo (aunque sea un tipo especial de datos llamado void, que viene siendo el equivalente a no devolver nada). En **Php** solo existen funciones, pero no están obligadas a devolver un valor, con lo que también tendríamos un procedimiento.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

6.- Validación y verificación de sistemas.

Verificación y validación son dos conceptos parecidos. De hecho, es común encontrarlos referidos a la misma idea. Pero, mientras que la **verificación** se refiere al **conjunto de actividades que aseguran que el software realiza una funcionalidad de manera correcta**, la **validación** se encarga de **asegurar que el software construido cumple con la funcionalidad requerida por el cliente**.

Ambos están englobados dentro un **contexto de análisis estático**, ya que no necesitan de la ejecución del programa.

La **validación** responde a la **pregunta** de si **se está haciendo el producto correcto**, es decir, se refiere a si cumple con los requisitos del cliente.

La **verificación** responde a si **se está haciendo el producto correctamente**, en otras palabras, comprobar que el producto es funcionalmente correcto y robusto.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

6.1.- Planificación.

Los **procesos** de validación y verificación **son caros**, por lo cual se hace recomendable una **planificación cuidadosa que permita obtener el máximo provecho y controlar los costes derivados**.

En circunstancias ideales, estos procesos deberían ser **introducidos en fases tempranas** del proceso de desarrollo, **estableciendo una lista de comprobación y** definiendo los posibles **planes de pruebas** para tener en cuenta la asignación de recursos.

Además, con estas acciones, se podrá estimar el calendario de pruebas que **permitirá una mejor visión general durante el proceso de desarrollo**.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

6.2.- Métodos formales de verificación.

Utilizan **técnicas matemáticas y lógicas** para comprobar que un programa cumple con su especificación. Se analiza la especificación para crear otra semánticamente equivalente; si se verifica esta última se da por validada la original.

Permite **verificar** que el **código del programa** es **coherente con su especificación**.

También resulta **útil para descubrir errores de programación y de diseño**.

Desventajas:

- Presentan un **gran coste** en forma de software especializado y expertos matemáticos.
- Son prácticamente **imposibles de aplicar en grandes desarrollos**, aunque siempre se puede reservar su aplicación a componentes críticos.

Los métodos formales pueden tener **aplicación en otras fases del ciclo de desarrollo** del software, tales como **análisis de requisitos y pruebas**.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

6.3.- Métodos automatizados de análisis.

Utilizan herramientas software que **escanean el código** de un programa en **busca** de posibles **errores**, comprobando **además que** el código **esté bien construido** de acuerdo con la sintaxis del lenguaje de programación utilizado.

Tres niveles de comprobación:

1. Comprobación de errores característicos:

- *Variables usadas antes de la inicialización.*
- *Variables declaradas pero no usadas.*
- *Variables asignadas dos veces pero nunca usadas entre ambas asignaciones.*
- *Asignaciones fuera de rango.*
- *Código inalcanzable.*
- *Bucles infinitos.*
- *Funciones y procedimientos no usados.*
- *Valor de retorno de función no almacenado.*
- *Errores de tipo de dato.*

2. Comprobación de errores definidos por el usuario

3. Comprobación de aserciones (dar por cierto)

7.- Pruebas de software.

Las pruebas software son el **conjunto de actividades destinadas a verificar de manera objetiva que se ha generado un software de calidad, libre de errores y que cumple con lo exigido.**

En este capítulo, se recorrerán los **diferentes tipos de prueba**, abordándose el proceso de **diseño de las mismas**, su **ámbito de aplicación** y las **herramientas software** que permiten su automatización.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.1.- Tipos.

Varias clasificaciones en función de la aplicación de diversos criterios:

- Según el **enfoque** utilizado:
 - Pruebas de tipo caja negra (black box testing).
 - Pruebas de tipo caja blanca (white box testing).
- Según la **necesidad de ejecución** del sistema a probar:
 - Pruebas estáticas (sin ejecutar el código).
 - Pruebas dinámicas (ejecutando el código).
- Según el **tipo de ejecución** de las pruebas:
 - Pruebas manuales.
 - Pruebas automáticas.
- Según el **ámbito** de las pruebas:
 - Pruebas unitarias.
 - Pruebas de integración.
 - Pruebas de sistema.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.2.- Ámbitos de aplicación.

El ámbito de aplicación de las pruebas **es el objetivo a probar durante la realización de las mismas**. En función del alcance de dichas pruebas se establecen tres niveles (o ámbitos) de aplicación, partiendo del nivel más específico para llegar al más general.

- **Módulo único: pruebas unitarias.** (módulos individuales, componentes, subprogramas, etc.)
- **Grupo de módulos: pruebas de integración.** (verificar la correcta interacción entre todos los componentes del sistema)
- **Sistema completo: pruebas de sistema.** (verifican el funcionamiento del sistema en su conjunto)

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.3.- Pruebas funcionales (BBT).

Estas pruebas **verifican un software usando su interfaz externa**. El sistema es visto como una caja negra de la que no se conoce su funcionamiento interno ni la estructura del sistema. Únicamente **basta con saber cuáles son las entradas que debe recibir la aplicación, así como cuáles son las correspondientes salidas**.

La **interfaz** tiene que estar **muy bien definida** para llevar a cabo estas pruebas correctamente.

Algunos de los métodos utilizados en el diseño de estas pruebas son:

- **Partición de equivalencia.**
- **Análisis de valores frontera.**
- **Prueba de arreglo ortogonal.**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.4.- Pruebas estructurales (WBT)

A diferencia del tipo caja negra, que implicaba una visión externa, estas pruebas **conllevan una visión interna**. Requieren de **conocimiento del código** para un correcto análisis de resultados. La persona encargada de efectuar estas pruebas elige **diversos valores de entrada, examinando el flujo del programa** y comprobando que se **devuelven los valores correctos**. Los casos de prueba derivados, siempre que se ejecuten de manera exhausta, **cumplirán** lo siguiente:

- *Garantizar que todas las rutas se revisan.*
- *Revisar todas las decisiones lógicas.*
- *Ejecutar todos los bucles con sus valores frontera.*
- *Revisar las estructuras de datos internas.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.5.- Diseño de pruebas (I).

Diseño de pruebas para caja negra se pueden usar los siguientes métodos:

- **Partición de equivalencia:**
 - Si requiere un rango: se especifica una clase de equivalencia válida (dentro del rango) y dos inválidas (fuera del rango).
 - Si requiere un valor específico: se define una clase válida (valor específico) y dos inválidas (otros valores).
 - Si requiere un booleano: se define una clase válida (verdadero) y una inválida (falso).
- **Análisis de valores frontera:**
 - Si se especifica un rango de valores entre a y b como condición de entrada, los casos de prueba se harán con los valores a y b.
- **Prueba de arreglo ortogonal:**
 - Si tenemos un programa con tres puntos de entrada que pueden tomar un valor del rango (1, 2, 3). Si se quiere realizar una prueba exhaustiva, habría que probar todas las combinaciones, que son 27 casos de prueba.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.5.- Diseño de pruebas (II).

Diseño de pruebas para caja blanca primer método:

- **Prueba de la ruta básica :**

Se crea un **grafo de flujo del programa**, Las **instrucciones** se representan por **nodos**, mientras que las **aristas** indican **el flujo** que puede seguir el programa.

Sobre el grafo se establece la **complejidad ciclomática**, que indicará el grado de complejidad lógica (se obtiene el numero de rutas independientes):

$$V(G) = E - N + 2$$

Siendo *E* el número de aristas del grafo y *N* el número de nodos.

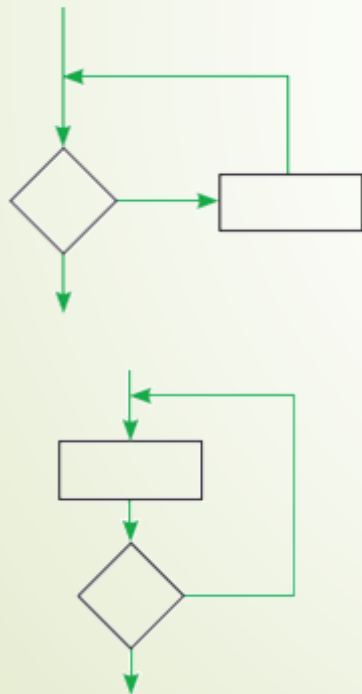
ejemplo

7.5.- Diseño de pruebas (III).

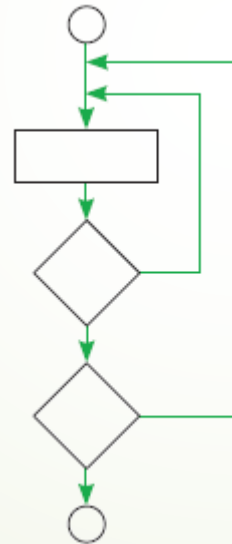
Diseño de pruebas para caja blanca segundo métodos:

- **Prueba de bucle:** se centra en los bucles, verificándolos. Distingue cuatro tipos:

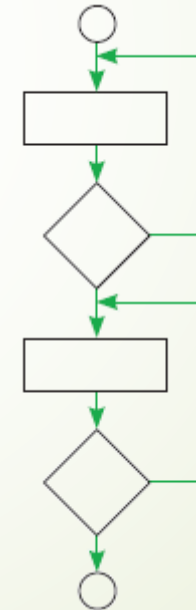
Bucles simples



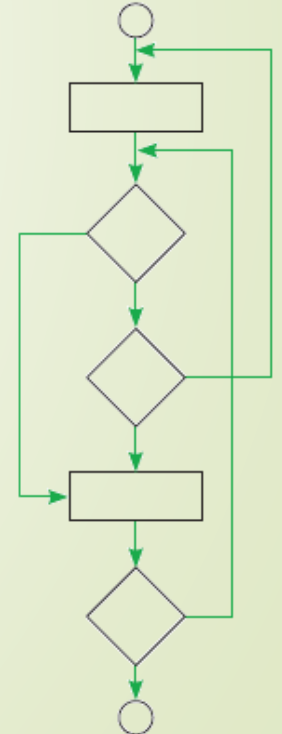
Bucles anidados



Bucles concatenados



Bucles NO estructurados



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.6.- *Comparativa. Pautas de utilización.*

El primer enfoque es una **visión externa** (pruebas de la caja negra), mientras que el segundo enfoque es una **visión interna** que conlleva revisar la estructura interna del programa (pruebas de la caja blanca). Ambas pruebas **se deben llevar a cabo para garantizar la calidad del producto**, utilizando una pauta general que se puede resumir de la siguiente manera:

1. Planificación y control
2. Análisis y diseño
3. Implementación y ejecución
4. Evaluar criterio de salida e informe
5. Cierre de pruebas

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.7.- Pruebas de componentes.

Misión: **verificar la correcta interacción entre los componentes de sistema.** Se parte de una representación del sistema en forma de árbol.

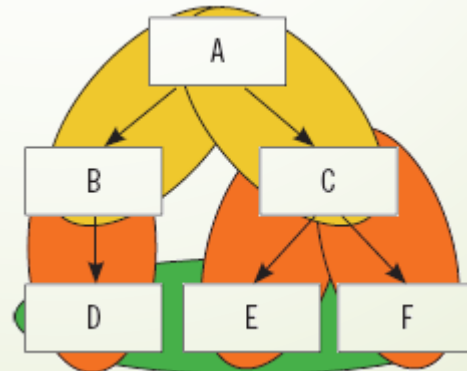
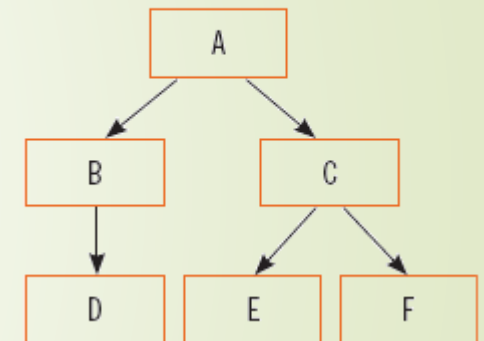
Existen cuatro estrategias para llevarlas a cabo:

- Big bang
- Incremental ascendente
- Incremental descendente
- Sándwich

Definición: Pruebas de regresión Pruebas que se realizan cuando se efectúa una modificación en el sistema, se agrega una nueva funcionalidad o se integra un nuevo módulo, con el fin de verificar el correcto funcionamiento del conjunto.

Ej. Incremental Ascendente

Componentes del sistema



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.8.- Pruebas del sistema.

Se corresponden a la validación del sistema, comprueba si el programa responde a los requisitos pedidos. Se necesita:

- **El software completo.**
- **Especificación de requisitos.**
- **Documentación de usuario.**

Se realizan en un **entorno real**. Las **más importantes** son las **pruebas de validación**, realizadas por usuarios finales o clientes. En presencia del equipo programador (**pruebas alfa**) o en ausencia del mismo (**pruebas beta**).

Entre los **aspectos que se tratan de verificar** se encuentran los siguientes:

- **Seguridad.**
- **Recuperación:** frente a errores, excepciones.
- **Rendimiento en condiciones extremas:** muchos usuarios, gran volumen de datos, muchas operaciones con base de datos.
- **Eficiencia:** capacidad para realizar o cumplir adecuadamente una función
- **Interacción con otro software.**
- **Compatibilidad.**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.9.- Automatización de pruebas. Herramientas.

Consiste en el *uso de un software especial que ejecuta pruebas de manera controlada, presentando resultados y comparando con los esperados.*

Generalmente, el software permite dos tipos de pruebas:

- **Pruebas manejadas por el código:** automatización de pruebas unitarias mediante casos de prueba.
- **Pruebas de interfaz de usuario:** permiten grabar acciones que realizaría un usuario sobre la interfaz del producto a evaluar.

Para **automatizar** el **desarrollo de pruebas**, se recurre al **uso de frameworks**, tales como JUnit (en entornos Java) y NUnit (en entornos .NET). Ambos forman parte de lo que se conoce como xUnit, que no es más que una agrupación de todos los frameworks Unit.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.10.- Estándares sobre pruebas de software (I).

Actualmente, se está desarrollando **el estándar ISO/IEC 29119**, el cual cubrirá áreas no consideradas por estándares anteriores. Objetivos: **unificar estándares, abarcar completamente el ciclo de vida** y ser **consistente con otros estándares ISO**.

Los estándares que se verán afectados serán los siguientes:

- **IEEE 829 Test Documentation.**
- **IEEE 1008 Unit Testing.**
- **BS 7925-1 Vocabulary of Terms in Software Testing.**
- **BS 7925-2 Software Component Testing Standard.**

El estándar **constará de cinco partes**, estando definidas así:

- **Parte 1: Definiciones y vocabulario.**
- **Parte 2: Proceso de pruebas.**
- **Parte 3: Documentación de pruebas.**
- **Parte 4: Técnicas de pruebas.**
- **Parte 5: Pruebas basadas en palabras clave (keyword-drive testing).**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

7.10.- Estándares sobre pruebas de software (II).

Como el nuevo estándar se encuentra en fase de desarrollo, nos centramos en el **IEE 829 Test Documentation**. 1ª versión 1983 y última 2008. Define los siguientes documentos:

- *Plan de pruebas*
- *Especificaciones de diseño*
- *Informe de ejecución:*
 - *Histórico de pruebas*
 - *Informe de incidente*
 - *Informe resumen de pruebas*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.- Calidad del software.

La calidad del software es el **conjunto de cualidades que lo caracterizan**. Pressman la define como “la concordancia con los requisitos funcionales y de rendimiento, con los estándares de desarrollo y con las características implícitas que se espera del software desarrollado profesionalmente”.

Esta **calidad es medible**, dependiendo del programa y del sistema sobre el que se ejecuta. Por ejemplo: no es lo mismo la calidad esperada de un software ocasional hecho para una única ejecución que la de un software que deba funcionar sin errores durante cinco años.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.1.- *Principios de calidad del software.*

Se pueden establecer unos aspectos generales sobre los que apoyar la calidad del software:

- *Corrección.*
- *Robustez.*
- *Eficiencia.*
- *Portabilidad.*
- *Integridad.*
- *Facilidad de uso.*
- *Verificabilidad.*
- *Extensibilidad.*
- *Reutilización.*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.2.- Concepto de métrica y su importancia en la medición de la calidad.

Una métrica es una medida de alguna propiedad del software. Proporciona un dato objetivo de aplicación durante los procesos de evaluación en el ciclo de desarrollo:

- **De complejidad.**
- **De calidad.**
- **De desempeño.**

Deben presentar las siguientes características:

- **Simples.**
- **Facilidad de cálculo.**
- **De naturaleza empírica.**
- **Objetivas.**
- **Independientes del lenguaje de programación elegido.**
- **Facilidad de uso para realimentación.**

La característica **más importante** de una métrica es la **realimentación**. De nada sirve obtener una medición sobre un aspecto del programa si no se pueden obtener **conclusiones que permitan realizar ajustes o modificaciones** en el caso de ser necesarias.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.3.- Métricas y calidad del software.

Dado que la finalidad general del software es ser funcional y de calidad, la **métrica proporciona medidas que permiten evaluar** esa **calidad de manera objetiva**.

Para controlar estas medidas, se establecen modelos de calidad. Según el ISO/IEC 8402, un **modelo de calidad** se define como “**el conjunto de características y las relaciones entre ellas que proveen la base para la especificación de los requisitos de calidad y la evaluación de la calidad**”.

Un modelo de calidad bien definido permite:

- **Una definición estructurada de los criterios usados para la evaluación.**
- **Una especificación de requisitos con relación a los criterios.**
- **La descripción de componentes en un marco común.**
- **La definición de métricas.**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.4.- Principales métricas en las fases del ciclo de vida software(I).

Métricas en el modelo de análisis

Intentan predecir el tamaño del sistema. Se obtiene un valor, partiendo de ciertos datos y aplicando alguna fórmula. Servirá de referencia para sucesivas planificaciones y revisiones:

- *Métricas basadas en la función*
- *Métrica bang*

Métricas de diseño

- *Métricas de diseño a nivel arquitectónico*
 - *Tamaño*
 - *Profundidad*
 - *Anchura*
- *Métricas de diseño a nivel de componentes*
 - *Métricas de cohesión*
 - *Métricas de acoplamiento*
 - *Métricas de complejidad*
- *Métricas de diseño de interfaz*

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.4.- Principales métricas en las fases del ciclo de vida software(II).

Métricas de codificación

Estas métricas ayudan durante la fase de codificación del programa, principalmente ofreciendo datos sobre la complejidad del código:

- **Complejidad ciclomática**
- **Otras métricas (programación estructurada)**
- **Otras métricas (programación orientada a objetos)**

Métricas para pruebas

Estas métricas se aplican durante la fase de prueba. Entre los datos que proporcionan, se encuentran medidas sobre el éxito de las pruebas, la cobertura de requisitos y la tasa de eliminación de errores.

- **Cobertura de funcionalidad**
- **Cobertura de funcionalidad**
- **Volatibilidad de requisitos**
- **Eficacia de revisión**
- **Eficiencia de eliminación de errores**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.6.- Estándar ISO/IEC 9126.

El ISO 9126 es un **estándar internacional** para la **evaluación de la calidad del software**. Está dividido en cuatro partes:

- **Modelo de calidad:** define las características que debe presentar un software de calidad, descomponiéndolas en subcaracterísticas:
 - **Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad, Portabilidad.**
- **Métricas externas:** encargadas de cuantificar la calidad externa del software. Miden las características anteriores.
- **Métricas internas:** miden la calidad interna del software en base a las subcaracterísticas definidas en la primera parte.
- **Métricas de calidad de uso:** miden si un producto cumple las necesidades especificadas por el usuario, distinguiendo entre defecto y no conformidad. Estas métricas solo pueden ser medidas en un entorno real:
 - **efectividad, productividad, seguridad, satisfacción.**

No conformidad y defecto La no conformidad se refiere al incumplimiento de un requisito, mientras que el defecto se refiere a una correcta implantación.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

8.7.- Otros estándares. Comparativa.


Aparte del ISO 9126, derivado de la familia ISO 9000, existe el estándar **ISO/IEC 25000**, de **muy reciente creación** y fruto de la **evolución del anterior** (al cual irá reemplazando poco a poco). Está compuesto por **cinco divisiones**:

1. ISO/IEC 2500n - **División de gestión de calidad**: define los modelos y términos comunes a toda la familia 25000.
2. ISO/IEC 2501n - **División de modelo de calidad**: representación de modelos detallados para calidad interna, externa y en el uso del producto software.
3. ISO/IEC 2502n - **División de medición de calidad**: definición de métricas para medir la calidad interna, externa y de uso.
4. ISO/IEC 2503n - **División de requisitos de calidad**: especificación de requisitos de calidad válidos durante el proceso de desarrollo.
5. ISO/IEC 2504n - **División de evaluación de calidad**: recomendaciones y guías válidas para el proceso de evaluación del software.

El ISO 25000 recibe el sobrenombre de SQuaRE (Software Product Quality Requirements and Evaluation).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.- *Herramientas de uso común para el desarrollo de software.*



En este punto se hará un repaso general a las **diversas herramientas** que tienen **utilidad** durante el **desarrollo del software**. Muchas de ellas agrupan varias funcionalidades en la misma herramienta, de complejidad más o menos variable, pudiendo ser **catalogadas como imprescindibles en función al desarrollo** que se quiera realizar.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.1.- Editores orientados a lenguajes de programación.

Editor: programa que **permite la creación (y edición) de archivos de texto**. Hay muchísimas alternativas, gratuitas, que facilitan enormemente el trabajo del programador. Entre algunas de las características, generalmente comunes, se encuentran:

- **Coloreado de sintaxis:** reconoce el lenguaje de programación usado en el archivo, coloreando el código según determine.
- **Edición de múltiples archivos:** despliegue en pestañas independientes. Muchas veces se ofrece la posibilidad de comparar dos archivos, indicando las diferencias.
- **Formateo de texto**, con sangrados y saltos de línea.
- **Conexiones con repositorios o FTP.**
- **Enlaces a otras herramientas** (compiladores, repositorios, etc.)

Entre los más populares, se encuentran **NotePad++** y **SublimeText 3**. El primero se encuentra disponible para Windows, mientras que el segundo, además, tiene versión para Linux y para OS/2.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.2.- Compiladores y enlazadores.

Un **programa** puede ser escrito en muchos lenguajes de programación, pero para su **ejecución** debe ser **traducido a un lenguaje entendible por el ordenador**. El **primer paso de la traducción lo realiza el compilador**, transformando el código en lo que se conoce como código en ensamblador.

Este último código **debe ser interpretado por el ensamblador**, con el fin de generar los archivos objeto. Estos archivos, generalmente, no son ejecutables y requieren de la actuación del enlazador.

El **enlazador** se encarga de **generar el ejecutable**, ligando los archivos objeto con otros archivos fuente o con funciones de una biblioteca (siempre que sea necesario).

El **ejemplo más clásico** de compilación lo ofrece el lenguaje de **programación C**, cuya secuencia se muestra en la siguiente imagen.

Proceso de compilación en C



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.3.- *Depuradores.*

Los depuradores (**debuggers**) son una de las **herramientas más útiles** para un programador. Integrados frecuentemente dentro de un entorno de desarrollo, un depurador **permite la ejecución del código línea a línea, presentando en todo momento información del estado de las variables** usadas en el programa y más datos de interés.

También ofrecen la **posibilidad** de **indicar puntos de interrupción**, que son lugares señalados dentro del código a partir del cual el programa entrará en modo depuración. Esto **resulta especialmente útil** para evitar partes del código que funcionan de manera correcta, yendo directamente a las líneas que interesa depurar.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.4.- Generadores de programas.

Los generadores de programas son **herramientas que crean software basado en algún lenguaje de programación sin necesidad de tener nociones del mismo**. El concepto es muy amplio, al igual que la forma de llevar a cabo esta generación de código. Por **ejemplo**, PHPRuner permite generar una interfaz plenamente funcional en PHP sobre una base de datos, partiendo de la misma. Otros, como Adobe Muse, habilitan al usuario para desarrollar un sitio web basado en la combinación de HTML5, CSS3 y JavaScript.

Sabía que...

Muchos IDE permiten la generación automática de código, sobre todo en lo referido a interfaz gráfica. Si se desea, por ejemplo, agregar un botón a una ventana, simplemente arrastrando el componente sobre la ventana contenedora se implementará todo el código (con el consiguiente ahorro de tiempo por parte del programador).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.5.- *De pruebas y validación de software.*

Estas herramientas **se encargan de automatizar pruebas unitarias y de otro tipo** (tales como pruebas de regresión, con el fin de comprobar si el programa se ajusta a los requisitos). Completan su funcionalidad facilitando la labor de **creación de informes** y **cotejamiento entre resultados obtenidos y resultados esperados**.

Las pruebas unitarias eran aquellas que se realizaban sobre un componente para verificar su funcionalidad. El **framework JUnit** permite la integración dentro de entornos de desarrollo con el fin de realizar estas pruebas.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.6.- Optimizadores de código.

Los optimizadores **actúan sobre el código objeto** generado por el ensamblador, **realizando una optimización del mismo**. La forma en que llevan a cabo su propósito es variada:

- **Reordenación de código.**
- **Reducción de tamaño.**
- **Optimización de bucles.**
- **Eliminación de redundancias.**
- **Reordenación de operaciones.**
- **Eliminación de asignaciones muertas (no utilizadas).**

El código **resultado** será **equivalente al original**, pero llevará a cabo su misión de manera **más eficiente y ocupará menos espacio**.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.7.- *Empaquetadores.*

Los empaquetadores son herramientas que **crean un paquete instalable a partir de un código**. Un paquete software **lleva**, aparte del **código compilado**, cierta información adicional que puede ser la **descripción del paquete**, su **versión** y sus **dependencias** (otros paquetes de los que depende).

Los paquetes están muy arraigados en la filosofía Linux, pues buena parte de la funcionalidad de este se obtiene instalando paquetes a través del potente gestor que incluyen las distribuciones Linux.

Sabía que...

El concepto de empaquetador también existe en Windows, pero quizá desde una perspectiva un poco más clásica. Normalmente, es una herramienta que permite crear menús de instalación (como el típico único archivo setup o install para realizar la instalación de un programa).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.8.- Generadores de documentación de software.

Un generador de software es una **herramienta destinada a la generación automática de documentación**, ya sea para el usuario final o para el programador.

9.9.- De distribución de software.

La distribución de software **consiste en proporcionar software a un usuario final**. Estas herramientas, dependiendo de la configuración adoptada, **pueden llegar a automatizar el proceso de instalación del nuevo software, llegando a estar completamente operativo en el ordenador con una mínima participación del usuario**.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.10.- Gestores y repositorios de paquetes. Versionado y control de dependencias (I).

Los gestores son una **colección de herramientas que abarcan todo lo relacionado con la vida de un paquete software**, desde su instalación hasta su posible eliminación. Un gestor de paquetes **normalmente va asociado a un repositorio**, que es un sitio especial en Internet que contiene las últimas versiones de una serie de paquetes.

El gestor de paquetes **verifica la versión de los paquetes que el usuario tiene instalados** en el sistema, ofreciendo la posibilidad de descargarlos (o haciéndolo automáticamente si ha sido programado para ello). Al mismo tiempo, **mantiene un control de dependencias**, con el fin de indicar los paquetes adicionales que hacen falta para el funcionamiento de uno en concreto.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.11.- Gestores de actualización de software (I).

Los gestores de actualización son pequeños programas que se instalan de manera complementaria a uno mayor. Su **misión** es **monitorizar la versión del software actual**, realizando cierta **acción cuando encuentren una versión superior** del programa al cual están asociados.

Como normal general, el **comportamiento del gestor de actualización** se puede definir de tres maneras diferentes:

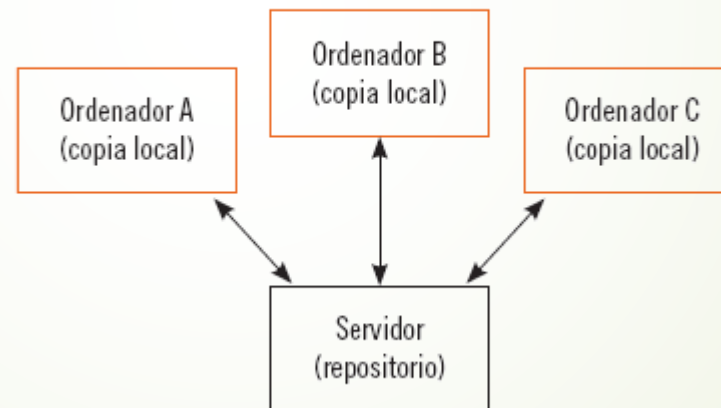
- **Permisos absolutos para instalar la última versión sin consultar al usuario.**
- **Preguntar al usuario antes de iniciar la actualización.**
- **No comprobar nunca actualizaciones.**

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.12.- De control de versiones.

Las herramientas de control de versiones **se utilizan para llevar el registro de todas las modificaciones realizadas en el código de un proyecto software, permitiendo restaurar cualquier versión anterior.** Actualmente, la filosofía es tener un repositorio en un servidor, al cual pueden acceder los desarrolladores para efectuar la edición del código del programa.

Filosofía general del control de versiones



UF1844: Desarrollo de aplicaciones web en el entorno servidor.

9.13.- *Entornos integrados de desarrollo (IDE) de uso común.*

Un IDE es una **herramienta** que **integra** muchas de las utilidades vistas en los puntos anteriores (generalmente **editor de código, compilación, depurador** y **automatización de pruebas**), ya sea de serie o mediante plugins.

Existen **multitud de IDE**, tanto **gratuitos** como de pago, incluso abarcando varios lenguajes de programación al mismo tiempo.

- **Visual Studio** está desarrollado por Microsoft.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

10.- Gestión de proyectos de desarrollo de software.

Se ofrecerá una visión general sobre la administración de proyectos de desarrollo, además de mostrar algunas herramientas útiles que facilitan esta labor.

10.1. Planificación de proyectos (I)

Es un aspecto de **suma importancia**. Una mala planificación desemboca habitualmente en un proyecto fracasado. *No es una ciencia exacta, que no se menosprecie, pero tampoco hay que obsesionarse con ella.*

El proceso para realizar **una correcta planificación** es el siguiente:

1. **Definir el ámbito del software.**
2. **Descomposición del problema en subproblemas.**
3. **Hacer estimaciones para cada subproblema.**
4. **Realizar la estimación global, considerando un factor de riesgo.**

La estimación requiere **especificar los recursos humanos y las herramientas software y hardware** de las que se dispone. **También** se indicará el **software reutilizable** que pueda ser de aplicación (componentes adquiridos o previamente desarrollados, módulos de otros proyectos, etc).

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

10.1. Planificación de proyectos (II)

Una vez realizada la estimación, hay que **considerar una planificación temporal** con lo siguiente:

- **Definir todas las tareas.**
- **Marcar las tareas que sean consideradas como críticas.**
- **Identificar el camino crítico.**

La **representación gráfica** de las tareas se realiza con Diagramas de Gantt, mientras que el cálculo del camino crítico se deriva de los Diagramas de Pert.

Diagrama de Gantt						
SEMANA	1	2	3	4	5	6
TAREA 1	[Barra naranja]					
Subtarea 11	[Barra verde]					
TAREA 2			[Barra naranja]			
Subtarea 21						
Subtarea 22						
TAREA 3				[Barra naranja]		
Subtarea 31				[Barra verde]		
Tarea 4					[Barra naranja]	[Barra naranja]
Subtarea 41						
Subtarea 42						
Subtarea 43						
Subtarea 44						

ejemplo

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

10.2.- Control de proyectos.

El **control** del avance del proyecto **se realiza en** los puntos marcados como **hito** en el calendario. En estas revisiones, **se repasa el estado del proceso** y pueden resultar muy útiles para encontrar problemas en fase temprana.

La revisión debe **comprobar** que **se está desarrollando un producto de calidad, ajustándose a los requisitos y cumpliendo lo indicado en la planificación**. También se puede establecer el uso de métricas, con el fin de obtener mediciones objetivas sobre la situación actual del proyecto.

Los hitos reciben el nombre de milestones. No solo sirven para indicar fechas de revisión en el calendario, sino que también puede referirse a acontecimientos importantes, como una entrega parcial.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

10.3.- *Ejecución de proyectos.*

La ejecución del proyecto es el **paso posterior a la planificación**, una vez conseguidos los recursos necesarios. Esta fase **engloba prácticamente todo el ciclo de vida del software**, aunque el análisis de requisitos se ha debido solapar ligeramente con la planificación del proyecto con el fin de poder plantear el producto.

Recuerde

Para el desarrollo de un software se usaba un modelo de ciclo de vida que definía las etapas y la interacción entre ellas con el fin de obtener un producto de calidad.

UF1844: Desarrollo de aplicaciones web en el entorno servidor.

10.4.- *Herramientas de uso común para la gestión de proyectos.*

Existen **multitud de herramientas software** que pueden ayudar al equipo de desarrollo durante un proyecto software. Entre las **funciones más típicas**, se encuentran la **gestión** de un **calendario**, con completas **administraciones de tareas y subtareas** y **definiciones de tiempo** de las mismas. Algunas herramientas, incluso, disponen de lugares de encuentro para **compartir documentación** o mantener una **videoconferencia**.