



U.T.1: El proceso de desarrollo de software.

[Fuente: Entornos de Desarrollo, Alicia Ramos, Ed. Garceta]

[Fuente: Análisis y diseño de Aplicaciones Informáticas de Gestión,
Mario Piattini, Ed. RA-MA]



Proceso de desarrollo de software

- ☐ Motivación.
- ☐ El software.
- ☐ Ciclo de vida del software.
- ☐ Fases del desarrollo de una aplicación.
- ☐ Lenguajes de programación.
- ☐ Herramientas.

Motivación

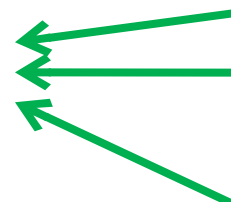
Información = Ppal. Activo de las empresas



desarrollo de SI \Leftarrow fuertes presiones
calidad,
productividad

~~¿Desarrollo artesanal suficiente?~~

\Rightarrow Disciplina de ingeniería
(sw. fiable, económico y eficiente)

 Gestión de calidad
Métodos (técnicas,
procesos, herramientas)
Gestión de proyectos

El software

- ❑ IEEE (729): *“el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación”.*
- ❑ Incluye:
 - ❑ *Instrucciones* que cuando se ejecutan, proporcionan la funcionalidad deseada.
 - ❑ *Estructuras de datos* que facilitan a las instrucciones manipular adecuadamente la información.
 - ❑ *Base de datos* que registran la información que maneja el sistema.
 - ❑ *Documentos* que describen el desarrollo, uso, instalación y mantenimiento de los programas.
 - ❑ Requiere tutorización => soporte al consumidor e instalación.

Características del software (frente al hardware)

- ❑ Más difícil de medir, validar, verificar:
 - ❑ Elemento lógico, no físico.
 - ❑ Desarrollado, no 'fabricado'.
- ❑ No se 'estropea', ¡pero se deteriora!
deterioro por cambios
- ❑ Mayoritariamente 'cerrado':
tradicionalmente, usado todo o nada
tradicionalmente, poco ensamblaje de componentes

Evolución del software

☐ **Década 1950-60:**

- ☐ “Software como un añadido”
- ☐ Continuos cambios en el hardware
- ☐ Desarrollo artesanal, a medida, sin métodos sistemáticos, planificación de tiempos ni de costes.

- ☐ Lenguajes de bajo nivel
- ☐ La misma persona es la que escribe, ejecuta, y depura el sw.

☐ **Década 1960-70:**

- ☐ Producción de software
- ☐ Primeras aplicaciones complejas
- ☐ BBDD
- ☐ Sistemas Multiusuario

- ☐ Los programas crecen en nº de líneas de código y se ve necesario el mantenimiento del software.
- ☐ “Crisis del software”

☐ **Década 1970-80:**

- ☐ Programación estructurada
- ☐ Modelo relacional
- ☐ Primeras etapas Ingeniería del Software
- ☐ Modelado de datos

- ☐ Redes LAN, WAN
- ☐ PC (Personal computer)
- ☐ El Hw empieza a bajar su coste
- ☐ El Sw crece y se venden centenas de miles de copias del software.

Evolución del software

☐ **Década 1980-90:**

- ☐ Programación OO
- ☐ 4GLs
- ☐ Tecnología de SGBDs, SOs
- ☐ Métodos estructurados

- ☐ Primeros métodos OO
- ☐ Tecnología CASE (1ª generación)

☐ **Década 1990-2000**

- ☐ Generalización POO
- ☐ Programación visual
- ☐ Tecnología de componentes
- ☐ Nuevas plataformas (Java, .NET)
- ☐ Análisis/Diseño OO

- ☐ UML (Unified Modeling Language, 1997)
- ☐ Patrones
- ☐ Tecnología CASE (2ª generación)
- ☐ Popularización de Internet

☐ **Década 2000-2010**

- ☐ Generalización comercio electrónico
- ☐ Web 2.0
- ☐ Desarrollo web

- ☐ Seguridad
- ☐ Arquitecturas basadas en servicios (SOA)
- ☐ Desarrollo opensource

Clasificación del software

☐ Una clasificación medianamente aceptada podría ser:

☐ Basado en el tipo de tarea que realiza

- ☐ De sistema
- ☐ De aplicación
- ☐ De programación

☐ Basado en el método de distribución

- ☐ Shareware
- ☐ Freeware
- ☐ Adware

- ☐ Multimedia
- ☐ De uso específico

☐ Teniendo en cuenta la licencia

- ☐ Software libre
- ☐ Software propietario
- ☐ Software de dominio público

Licencias de software

- ❑ Contrato que se establece entre el desarrollador de software y el usuario.
- ❑ El desarrollador o propietario de los derechos de explotación elige la licencia según la cual se distribuye el software.
 - ❑ Software libre y de código abierto
 - ❑ Software propietario
 - ❑ Software de dominio público

Licencias de software libre

- ❑ Libertad de usar el programa con cualquier fin y número de equipos
- ❑ Libertad de estudiar el funcionamiento del programa y adaptarlo a necesidades específicas
- ❑ Libertad de distribuir copias a otros usuarios (con o sin modificaciones)
- ❑ Libertad de mejorar el programa y de hacer públicas las modificaciones.

Licencias de software libre

- ❑ Free Software Foundation

- ❑ ‘The GNU General Public License is a free, copyleft license for software and other kinds of works’.

[<http://www.gnu.org/>]

- ❑ The Open Source Initiative

- ❑ Licencia Creative Commons

- ❑ No es la más recomendada para productos software.

Software propietario

- ❑ Se distribuye en formato binario, sin posibilidad de acceso al código fuente.
- ❑ Prohíbe todas o alguna de las formas de distribución que incluyan:
 - ❑ Redistribución del software
 - ❑ Modificación
 - ❑ Copia
 - ❑ Uso en varias máquinas simultáneamente
 - ❑ Transferencia de titularidad
 - ❑ Difusión de fallos y errores, etc.

Software de dominio público

- ❑ Carece de licencia
- ❑ Posiblemente se desconoce el autor
- ❑ Incluso cabe la posibilidad de desarrollar una licencia propietaria sobre la base de un código de dominio público

Ejercicio: Consulta el siguiente enlace donde se listan licencias existentes compatibles con la GPL

www.gnu.org/licenses/license-list.es.html#SoftwareLicenses

Evolución en el desarrollo de software

□ Práctica:

Investiga sobre el concepto '*Crisis del software*'

- En qué momento aparece
- A qué se debe
- Consecuencias que se derivan.

Evolución en el desarrollo de software

- ❑ Con el avance del hardware, necesidad de aplicaciones más complejas
 - ⇒ Se produjo un cambio en la relación coste hardware/software
- ❑ Problemas “tradicionales” del desarrollo:
 - ❑ Incapacidad para estimar tiempo, coste y esfuerzo para el desarrollo de un producto software.
 - ❑ Falta de calidad del producto software.

Evolución en el desarrollo de software

- ❑ Estas preguntas llevaron a la aparición y adopción paulatina de la *Ingeniería del Software* en el desarrollo:
 - ❑ ¿Por qué lleva tanto tiempo terminar los programas?
 - ❑ ¿Por qué es tan elevado su coste?
 - ❑ ¿Por qué no podemos encontrar todos los errores antes de entregar el software a nuestros clientes?
 - ❑ ¿Por qué nos resulta difícil constatar el progreso conforme se desarrolla el software?

Algunas causas

- ❑ Naturaleza “no física” de la programación.
- ❑ El software es la parte más ‘maleable’ del sistema.
- ❑ Problemas de comunicación con los clientes.
- ❑ Problemas derivados de la intervención de grupos.
- ❑ Problemas de gestión.

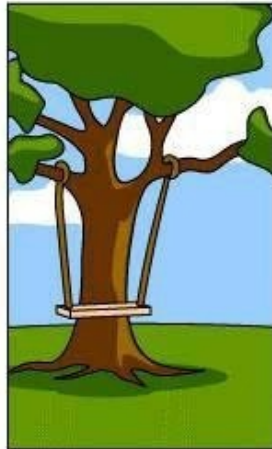
Planificaciones optimistas, plantillas poco cualificadas...

- ❑ Poco esfuerzo en análisis y diseño.
- ❑ Difusión limitada de las nuevas técnicas, métodos y herramientas.

El problema de la comunicación



Cómo el cliente lo explica



Cómo el líder del proyecto lo entiende



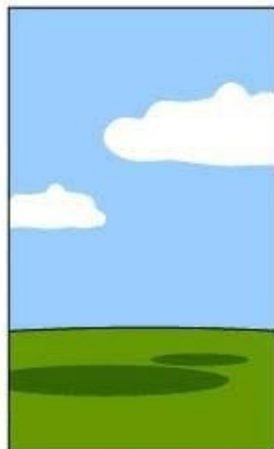
Cómo el analista lo diseña



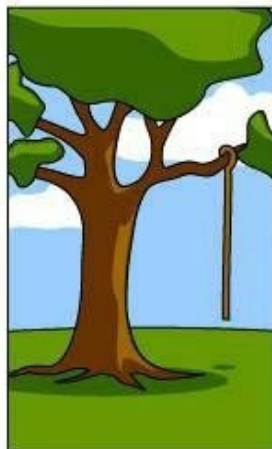
Cómo el programador lo escribe



Cómo el asesor lo describe



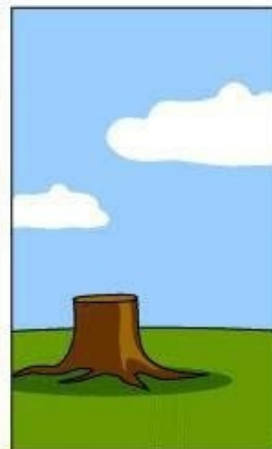
Cómo se documenta el proyecto



Qué aplicaciones se instalan



Cómo se factura al cliente



Así se le dará soporte



Lo que el cliente realmente necesitaba

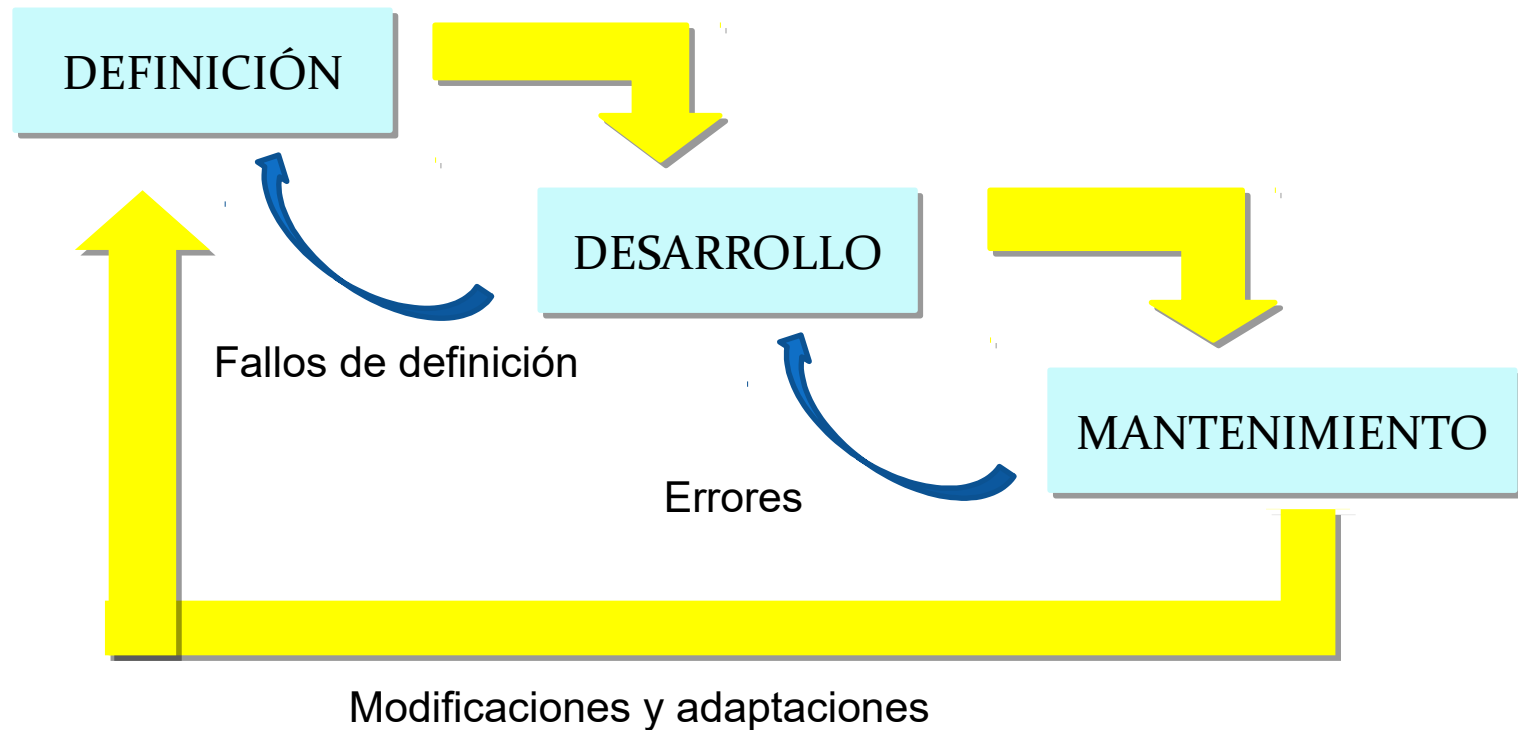
La ingeniería del software

- ❑ Término que aparece en 1968
- ❑ La producción de programas debe abordarse como una ingeniería más.
- ❑ (Boehm) La Ingeniería del Software es la aplicación práctica y sistemática del conocimiento científico a:
 - ❑ la producción de programas correctos, que se desarrollan a tiempo y dentro de las estimaciones de presupuesto,
 - ❑ y a la correspondiente documentación para desarrollarlos, usarlos y mantenerlos.

Más definiciones de ISW

- “Una disciplina que comprende todos los aspectos de la producción de software, desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza” (Sommerville 2002).

Visión general del proceso de ISW



En la práctica, no es secuencial: ha de ser **iterativo e incremental** ('procesos', no 'fases')

Concepto de ciclo de vida

“Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software”

IEEE 1074

“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 de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso”

ISO 12207-1

Ciclo de Vida

Ciclo de Vida \neq Ciclo de desarrollo

Desde el análisis
hasta la entrega al

usuario



Toda la vida del sistema:

Desde su concepción hasta el fin de su uso



Etapas del ciclo de vida

- ❑ Con independencia del área de aplicación, tamaño o complejidad del proyecto, el desarrollo de cualquier sistema se encontrará al menos en uno de los siguientes 'procesos' genéricos:
 - ❑ Definición ~ análisis (del sistema, del software)
 - ❑ Desarrollo ~ diseño, codificación y prueba
 - ❑ Mantenimiento
- ❑ Cada etapa tiene como entrada uno o varios documentos de la etapa anterior.

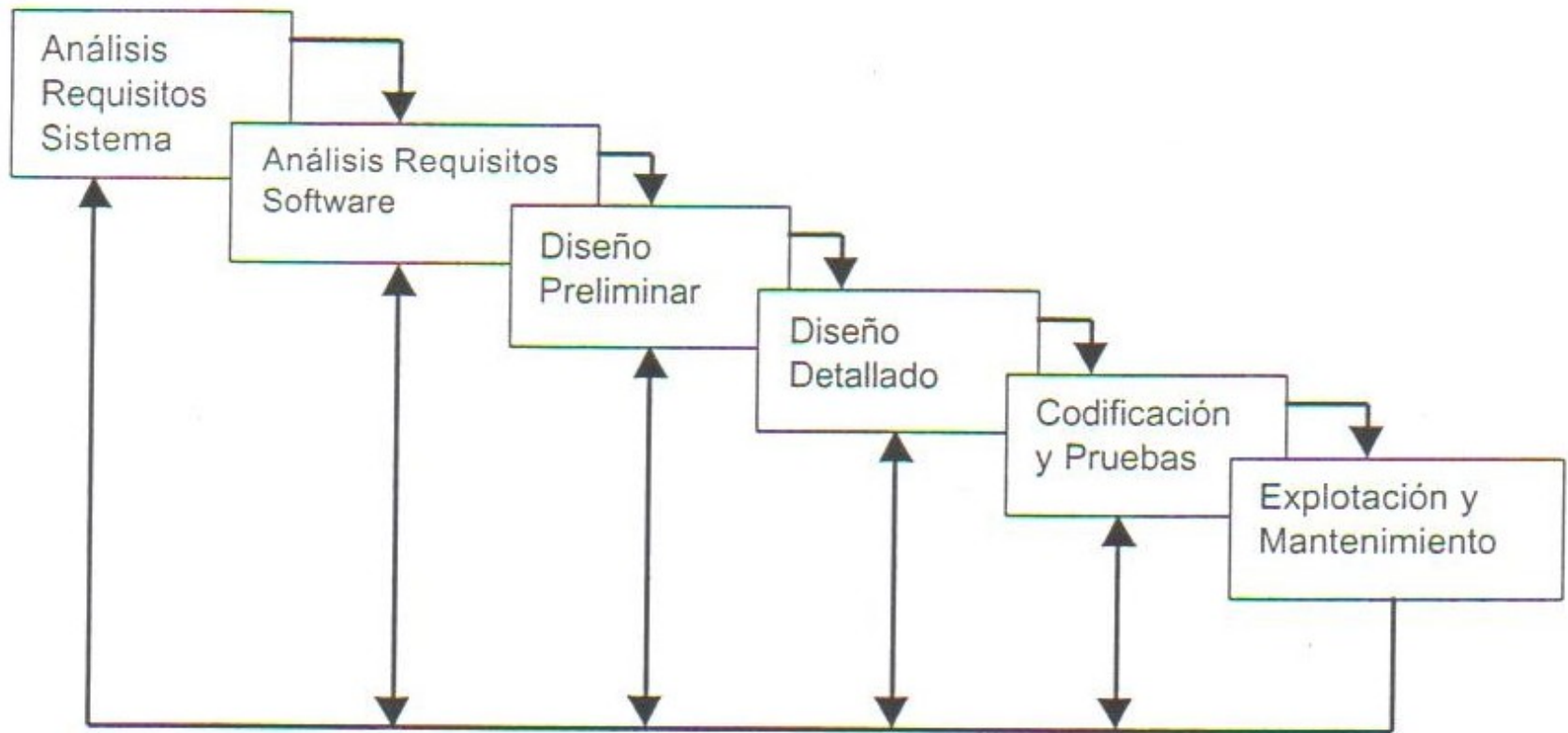


MODELOS DE CICLO DE VIDA

Modelo en cascada

- ❑ Primer modelo empleado (1970)
- ❑ También denominado
“ciclo de vida clásico” o *“paradigma clásico”*
- ❑ Ejecución secuencial de una serie de fases
- ❑ Cada fase empieza cuando termina la fase anterior.
- ❑ Para pasar de una fase a otra es necesario conseguir todos los objetivos de la etapa previa.
- ❑ Al final de cada fase el personal técnico y los usuarios revisan el progreso del proyecto.

Modelo en cascada



Modelo en cascada. A favor

- ❑ Es “sencillo” controlar qué productos se deben generar durante el desarrollo del proyecto
- ❑ Ayuda a prevenir que se sobrepasen las fechas de entrega y los costes esperados.
- ❑ La calidad del producto resultante es alta.

Modelo en cascada. Críticas

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

Modelo en cascada: Recomendación

- ❑ Proyecto similar a otros terminados con éxito
- ❑ Requisitos estables y bien comprendidos.
- ❑ No hay necesidad de versiones intermedias.
- ❑ Para un proyecto corto de p.ej. dos meses, se puede usar un ciclo de vida en cascada (Larman 2003)
 - ⇒ Las dificultades aparecen cuando la escala de tiempo se alarga

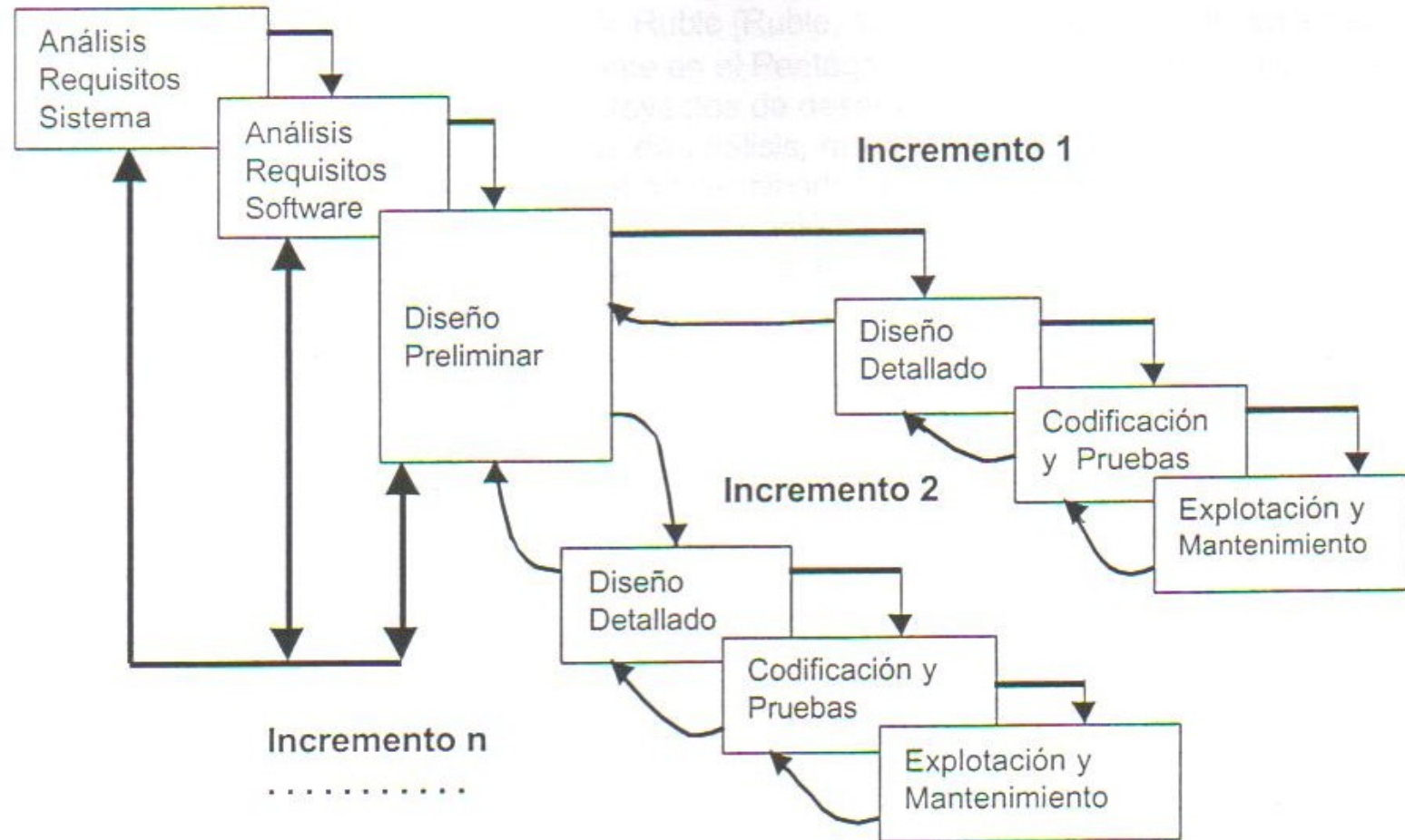
Modelos Evolutivos

- ❑ Asumen que los requisitos del producto cambien durante el desarrollo.
- ❑ Se entregan diferentes versiones del producto, cada vez más completas, hasta llegar al producto final.
- ❑ Los más conocidos:
 - ❑ Iterativo incremental
 - ❑ Modelo en espiral

Modelo incremental

- ❑ Utiliza varios ciclos en cascada realimentados aplicados repetidamente.
- ❑ Se va creando el sistema software añadiendo componentes funcionales al sistema (incrementos).
- ❑ En cada paso se actualiza el sistema con nuevas funcionalidades o requisitos (refinamiento del sistema).

Modelo incremental



Modelo incremental. A favor

- ❑ No hay necesidad de un conjunto exhaustivo de requisitos, especificaciones y diseños al principio.
- ❑ Permite la entrega temprana de partes operativas del software.
- ❑ Las entregas previas facilitan la realimentación de las siguientes versiones.

Modelo incremental. Críticas

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

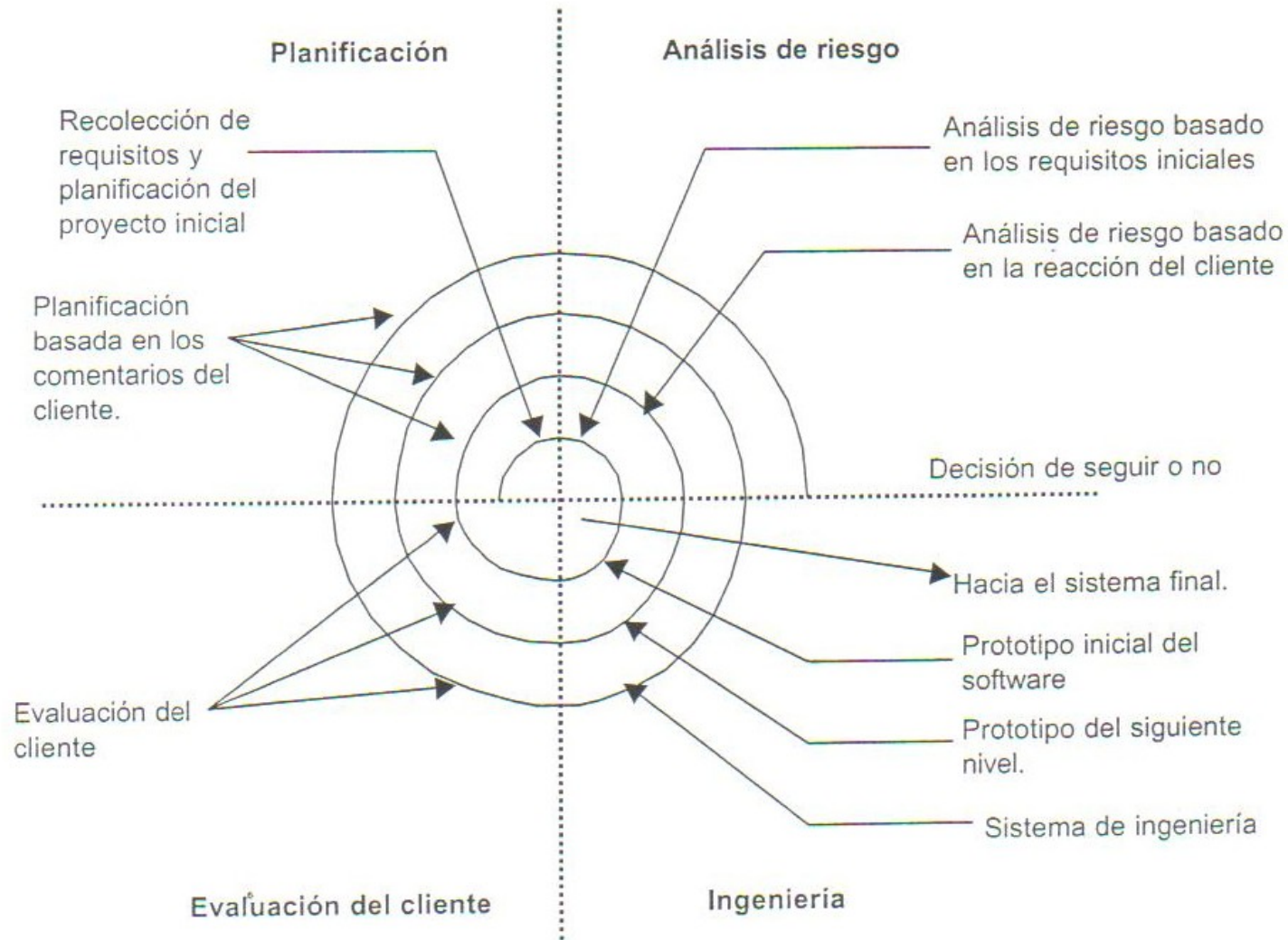
Modelo incremental. Recomendación

- ❑ Se ajusta a entornos de alta incertidumbre, donde los requisitos no están completos o pueden cambiar
- ❑ Se están probando e introduciendo nuevas tecnologías.

Modelo en espiral

- ❑ Combina el modelo en cascada con el modelo evolutivo de construcción de prototipos.
- ❑ Consta de una serie de ciclos en los que se desarrolla una parte del software.
- ❑ Cada ciclo está formado por cuatro fases:
 - ❑ Determinar objetivos, alternativas y restricciones
 - ❑ Análisis del riesgo y evaluación de alternativas
 - ❑ Desarrollar y probar la solución elegida
 - ❑ Evaluación del producto obtenido/Planificación de la siguiente etapa
- ❑ Al finalizar cada ciclo obtenemos una versión mejorada del software (prototipo) con respecto al ciclo anterior.

Modelo en espiral



Modelo en espiral. A favor

- ❑ No requiere definición completa de requisitos al principio.
- ❑ Análisis del riesgo en todas las etapas.
- ❑ Reduce los riesgos del proyecto.
- ❑ Incorpora objetivos de calidad.

Modelo en espiral. Críticas.

- ❑ Difícil evaluar los riesgos
- ❑ El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
- ❑ Necesidad de expertos en evaluación de riesgos para identificar y manejar las fuentes de riesgos de un proyecto.

Modelo en espiral. Recomendación.

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



FASES DEL DESARROLLO

Fases del desarrollo

- ❑ Antes de desarrollar un proyecto software se elige un modelo de ciclo de vida según las características del mismo.
- ❑ Todos los modelos siguen, en una u otra forma, una serie de etapas generales:
 - ❑ Análisis
 - ❑ Diseño
 - ❑ Codificación
 - ❑ Pruebas
 - ❑ Documentación
 - ❑ Explotación
 - ❑ Mantenimiento

Análisis

- ❑ ¿Qué debe hacer el sistema?
 - ❑ funcionalidad del sistema
 - ❑ información que ha de manejar
 - ❑ necesidades de rendimiento
 - ❑ restricciones de diseño
 - ❑ interfaces del sistema con los usuarios y con otros sistemas
 - ❑ criterios de validación
- ❑ Documentos de requisitos del sistema (SyRS, System Requirements Specification) (en su caso) y del software (SRS, Software Requirements Specification)

Análisis

- Una comunicación fluida cliente / desarrolladores es muy importante en esta etapa.
- Técnicas de recogida de requisitos:
 - Entrevistas
 - Brainstorming
 - Prototipos
 - Casos de uso
- Técnicas para representar los requisitos:
 - Diagrama de Flujos de Datos (DFD) [Fig 1.9]
 - Diagramas de flujo de control (DFC)
 - Diagramas de transición de estados (DTE)
 - Diagrama Entidad/Relación (DER)
 - Diccionario de Datos (DD)) [Pág 15]

Diseño

- ❑ ¿Cómo construir el sistema?
 - ❑ Se traducen los requisitos a una representación del software.
 - ❑ Se diseñan las estructuras de datos y los programas
 - ❑ cómo se caracterizan las interfaces
 - ❑ Principalmente hay dos tipos de diseño:
 - ❑ Diseño Estructurado
 - ❑ Diseño Orientado a Objetos

Diseño

- ❑ El diseño estructurado está basado en el flujo de los datos a través del sistema.
- ❑ Genera los siguientes documentos:
 - ❑ Diseño de datos
 - ❑ Diseño arquitectónico
 - ❑ Diseño de la interfaz
 - ❑ Diseño procedimental (a nivel de componentes)

[Figura 1.11]

Diseño

- Herramientas gráficas empleadas en el diseño estructurado:
 - Diagramas de flujo
 - Diagramas de cajas
 - Tablas de decisión
 - Pseudocódigo

Ejemplo tabla de decisión

[pág 20]

Diseño

- Herramientas gráficas empleadas en el diseño estructurado:

Ejemplo tabla de decisión

Reglas

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 compra	✓	✓		
Aplicar 3% descuento sobre el importe compra	✓		✓	
Calcular factura	✓	✓	✓	✓

Diseño

- Herramientas gráficas empleadas en el diseño estructurado:

Ejemplo de pseudocódigo

Inicio

Abrir Fichero

Leer Registro del Fichero

Mientras no sea Fin de Fichero Hacer

Procesar Registro leído

Leer Registro del Fichero

Fin mientras

Cerrar Fichero

Fin

Diseño

- ❑ En el diseño orientado a objetos el sistema se entiende como un conjunto de objetos y eventos.
- ❑ El diseño orientado a objetos define 4 capas de diseño:
 - ❑ Subsistema
 - ❑ Clases y Objetos
 - ❑ Mensajes
 - ❑ Responsabilidades
- ❑ La herramienta más utilizada para realizar el diseño OO es el lenguaje de modelado UML.

Codificación

- ❑ El programador recibe las especificaciones del diseño y las transforma en instrucciones del lenguaje de programación.
- ❑ Utilizar normas de programación y estilo claras y homogéneas

[http://
www.um.es/docencia/vjimenez/ficheros/practicas/C
onvencionesCodigoJava.pdf](http://www.um.es/docencia/vjimenez/ficheros/practicas/Con convencionesCodigoJava.pdf)

Codificación: Normas de estilo en java

- ❑ Extensiones para ficheros fuente **.java** y compilados **.class**
- ❑ Cada fichero debe contener una sola clase pública y debe ser la primera. Además:
 - ❑ Clases privadas e interfaces relacionadas a continuación.
- ❑ Organización de los ficheros de clases:
 - ❑ Comentarios
 - ❑ Sentencias *package* e *import* en este orden.
 - ❑ Declaraciones de clases e interfaces:
 - ❑ Sentencia class o interface
 - ❑ Variables estáticas: públicas, protegidas y luego privadas
 - ❑ Variables de instancia
 - ❑ Constructores.
 - ❑ Métodos

Codificación: Normas de estilo en java

□ Indentación

- 4 espacios
- longitud líneas de código ≤ 80 caracteres
- longitud comentarios ≤ 70 caracteres
- salto de línea después de comas o antes de operador y alinear con la anterior.

Codificación: Normas de estilo en java

- ❑ Los comentarios deben contener sólo información relevante.
 - ❑ Comentarios de documentación (delante del objeto documentado)
 - ❑ Comentarios de implementación (al lado de la línea comentada)

Codificación: Normas de estilo en java

❑ Declaraciones:

- ❑ Una variable por línea.
- ❑ Inicializar las variables locales en la declaración
- ❑ No separar el nombre del método de los paréntesis
- ❑ La llave de apertura del método se escribe en la misma línea que el nombre del mismo.
- ❑ La llave de cierre se coloca en una línea aparte y en la misma columna que el inicio del bloque
- ❑ Los métodos se separan por una línea en blanco

Codificación: Normas de estilo en java

□ Sentencias:

- Cada línea contiene una sentencia.
- Los bloques de sentencias deben estar sangrados con respecto a la sentencia que lo genera.
- Esta norma se aplica también a bloques condicionales y bucles.

Codificación: Normas de estilo en java

❑ Separaciones:

- ❑ Dos líneas en blanco entre las definiciones de clases e interfaces
- ❑ Una línea en blanco entre definiciones de métodos, definición de variables locales y primera instrucción dentro de un método, antes de un comentario y entre secciones dentro de un método.
- ❑ Un carácter en blanco entre una palabra y un paréntesis, después de una coma y en las expresiones del for.

Codificación: Normas de estilo en java

❑ Nombres:

- ❑ Los **paquetes** en minúscula y puede contener puntos. *java.io*
- ❑ Las **clases** e **interfaces** utilizan sustantivos con la primera letra de cada palabra en mayúscula si utiliza más de una y sin separación. *public class HiloServidor{ }*
- ❑ Los **métodos** se nombran en minúscula con verbos en infinitivo aplicando la regla anterior si contiene varias palabras salvo para la primera que va toda en minúscula. *ejecutar(), asignarDestino()*
- ❑ Las **variables** deben tener nombres cortos, en minúscula y siguen la regla de los métodos para nombres compuestos. *sumaTotal.*
- ❑ Las **constantes** se escriben en mayúscula con guión bajo para unir nombres compuestos. *MAX_VALOR*

Codificación

- ❑ Los entornos de desarrollo incorporan opciones para ayudar a formatear el código fuente. Eclipse:

menú Source -> Format con el fichero abierto

- ❑ El código fuente es compilado e interpretado para obtener el código ejecutable final.
- ❑ Simultáneamente a la producción de código se habrán generado los manuales necesarios para ejecutar la fase de prueba y explotación.
 - ❑ Manual técnico y de referencia
 - ❑ Manual de usuario

Codificación

```
public class Ejemplo{  
    public static void main ( String[] args) {  
        int suma = 0;  
        int contador = 0;  
        while (contador < 10) {  
            contador++;  
            suma = suma + contador;  
        }  
        System.out.println( "Suma => " + suma );  
    }  
}
```

```
public class  
Ejemplo {  
    public static  
  
    void main ( String[] args) {  
  
        int suma = 0; int contador = 0;  
        while (contador < 10)  
        {  
            contador++;  
            suma = suma + contador; }  
  
        System.out.println("Suma => "  
+ suma);  
    }  
}
```

Prueba

- ❑ Planificar y diseñar pruebas que detecten errores en el software ya disponible.
- ❑ Se buscan errores de codificación, diseño y especificación.
- ❑ Actividades en la etapa de prueba
 - ❑ Verificación: ¿El producto se construye correctamente?
 - ❑ Validación: ¿El producto se ajusta a los requisitos?
- ❑ Un ***caso de prueba*** es un documento que describe los valores de entrada, salida esperada y condiciones previas para una prueba.

Prueba



Figura 1.16. Flujo de proceso para probar el software.

Prueba

- ❑ Técnicas del diseño de casos de prueba:
 - ❑ Caja blanca: Validan los detalles procedimentales del código
 - ❑ Caja negra: Validan los requisitos funcionales.

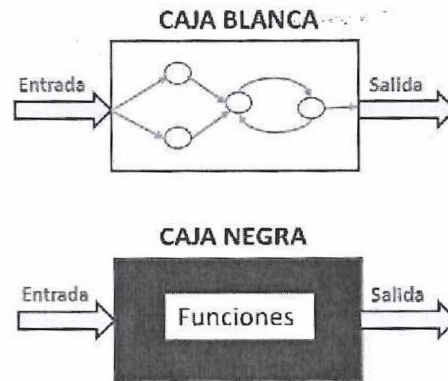


Figura 1.17. Pruebas de caja blanca y negra.

Factores de calidad del software

(internos y externos)

☐ **Correcto**

- ☐ Se ajusta a las especificaciones dadas por el usuario.

☐ **Fiable**

- ☐ Capacidad de ofrecer los mismos resultados bajo las mismas condiciones.

☐ **No Erróneo**

- ☐ No existe diferencia entre los valores reales y los calculados

☐ **Eficiente**

- ☐ Utilización óptima de los recursos de la máquina.

☐ **Robusto**

- ☐ No poseer un comportamiento catastrófico ante situaciones excepcionales (tolerante a fallos).

Factores de calidad del software

(internos y externos)

☐ ***Portable***

- ☐ Capaz de integrarse en entornos distintos con el mínimo esfuerzo.

☐ ***Adaptable (extensible)***

- ☐ Capaz de modificar alguna función sin que afecte a sus actividades

☐ ***Inteligible***

- ☐ Diseño claro, bien estructurado y documentado.

☐ ***Reutilizable***

- ☐ El software puede ser usado con facilidad en nuevos desarrollos.

Documentación

- ❑ Son todos los documentos que describen un sistema software.
- ❑ Los documentos relacionados con un proyecto software proporcionan:
 - ❑ Medio de comunicación para el equipo de desarrollo.
 - ❑ Repositorio de información para el mantenimiento.
 - ❑ Apoyo para la gestión del presupuesto y calendario.
 - ❑ Manual de usuario y administración.

Documentación

- ❑ En general encontramos dos clases de documentación:
 - ❑ Documentación del proceso
 - ❑ Productos obtenidos en las diferentes etapas del desarrollo
 - ❑ Documentación del producto
 - ❑ Documentación que describe el sistema desde un punto de vista técnico
 - ❑ Documentación orientada a los usuarios finales del producto.

Documentación

- “*IEEE 1063-2001 Standard for Software User Documentation*” propone una estructura para el manual de usuario



Descarga del site de clase el documento descriptivo

Explotación

- ❑ Instalación y puesta en marcha del producto software en el entorno de trabajo del cliente.
- ❑ Incluye definir un procedimiento para registrar, solucionar y hacer seguimiento de los problemas encontrados durante la ejecución en el entorno de trabajo.

Mantenimiento

- ❑ Comienza una vez construido el sistema, cuando se pone en explotación.
- ❑ Se centra en el **cambio**.
- ❑ El software es sometido a reparaciones y modificaciones cada vez que se detecta un fallo o se necesita cubrir una nueva necesidad de los usuarios.
- ❑ Recae el mayor porcentaje del coste de un sistema.

Mantenimiento

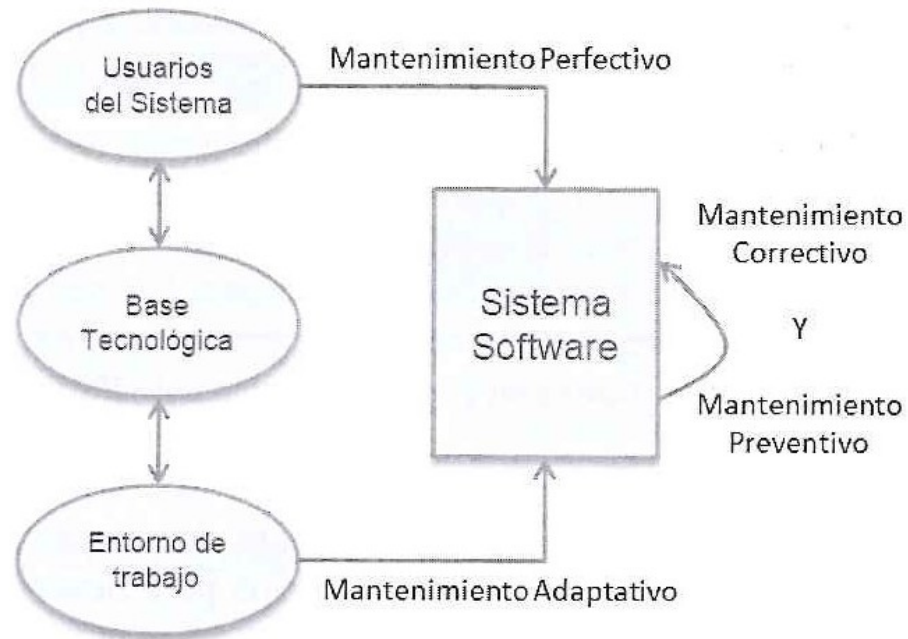


Figura 1.19. Tipos de mantenimiento del software.

Tipos de mantenimiento

- ❑ **Correctivo:** Modificaciones debidas a un mal diseño.
- ❑ **Perfectivo:** Modificaciones destinadas a mejorar la eficiencia, o recoger nuevas funcionalidades no expresadas en la fase de definición del sistema.

Tipos de mantenimiento

- ❑ **Adaptativo:** Modificaciones orientadas a incorporar los cambios del entorno externo (modificaciones en la legislación, CPU, SO, las reglas de negocio, etc.)
- ❑ **Preventivo:**
 - ❑ El software se deteriora con los cambios
 - ❑ Modificaciones para corregir, adaptar y mejorar más fácilmente el producto (**reingeniería del software**)



LENGUAJES DE PROGRAMACIÓN

Concepto de programa

- ❑ Conjunto de instrucciones escritas en un lenguaje de programación que aplicadas sobre un conjunto de datos resuelven un problema o parte del mismo.
- ❑ Las instrucciones de un programa se cargan en la memoria principal y la CPU se encarga de ejecutarlas.

Concepto de lenguaje de programación

- ❑ Conjunto de los siguientes elementos:
 - ❑ *Vocabulario* o alfabeto: Conjunto de símbolos permitidos
 - ❑ *Sintaxis* o reglas para combinar los símbolos del alfabeto
 - ❑ *Semántica* o reglas que dan significado a las combinaciones de símbolos.

Clasificación de los lenguajes de programación

- ❑ Según su nivel de abstracción
 - ❑ Lenguajes de bajo nivel
 - ❑ Lenguajes de nivel medio
 - ❑ Lenguajes de alto nivel
- ❑ Según la forma de ejecución:
 - ❑ Lenguajes compilados
 - ❑ Lenguajes interpretados

Clasificación de los lenguajes de programación

- Según el paradigma de programación:
 - Lenguajes imperativos
 - Lenguajes funcionales
 - Lenguajes lógicos
 - Lenguajes estructurados
 - Lenguajes orientados a objetos.

Obtención de código ejecutable

- ❑ El proceso de compilación se compone de dos programas, el compilador y el enlazador:

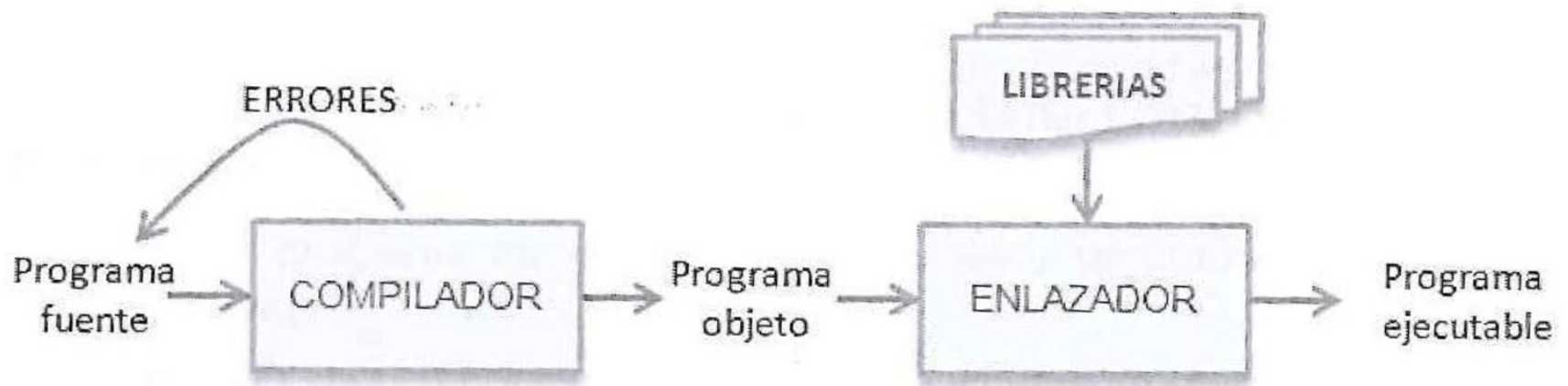


Figura 1.32. Proceso de compilación.

Obtención de código ejecutable

- ❑ El compilador pasa por varias etapas hasta generar el código objeto:

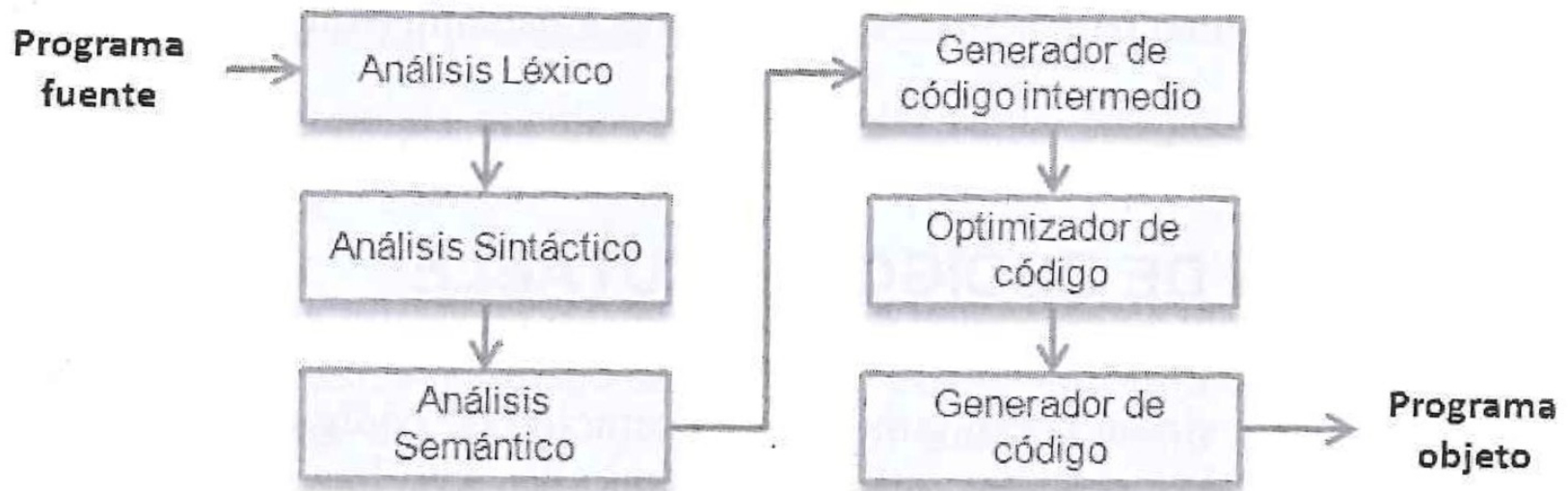


Figura 1.33. Fases de un compilador.

Máquinas virtuales

- ❑ Aplicaciones software que simulan el funcionamiento de una máquina real.
 - ❑ Máquinas virtuales de sistema
 - ❑ Máquinas virtuales de proceso

Máquinas virtuales

- ❑ La máquina virtual de Java (JVM)
 - ❑ Los programas compilados en lenguaje Java se pueden ejecutar en cualquier plataforma.
 - ❑ El código generado por el compilador se ejecuta en una máquina virtual

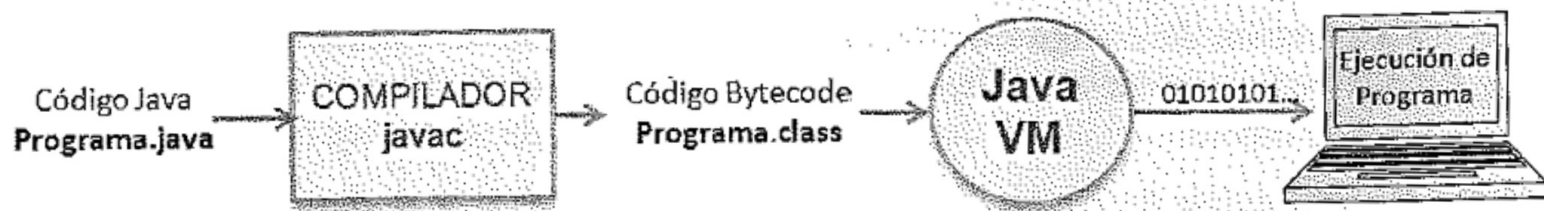


Figura 1.34. Compilación y ejecución de un programa Java.

Máquinas virtuales

- ❑ La máquina virtual de Java (JVM)

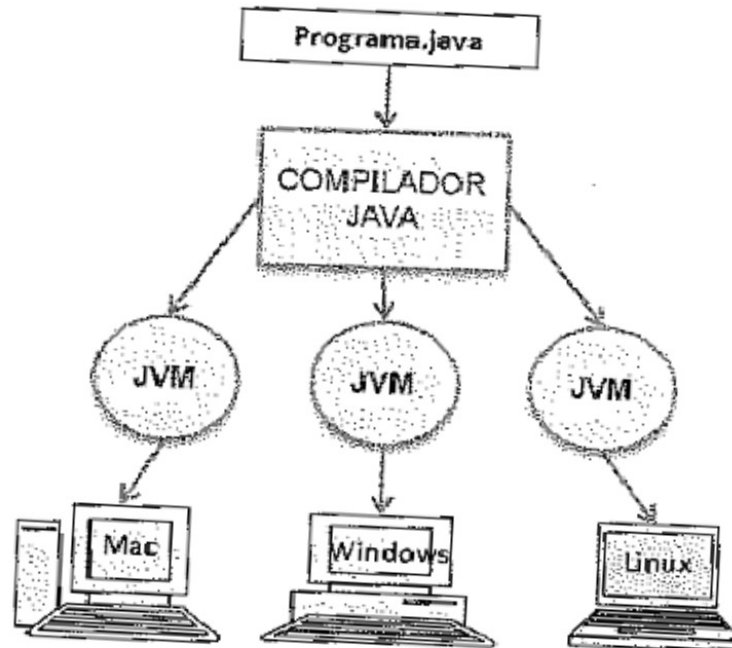


Figura 1.35. Ejecución de un programa Java en distintas plataformas.