2.3. Códigos fuente, objeto y ejecutable.

Válido para lenguajes compilados como C, C++, Java…

-Código fuente: Archivo de texto escrito en un lenguaje de programación.
Se puede editar.
Archivo binario ejecutable.
(.exe)

-Código objeto(no ejecutable): Archivo binario no ejecutable.
(.o)
No necesita ser compilado.

-Código ejecutable: Archivo binario ejecutable.

En lenguajes interpretados no existe código objeto ni binario, solo código fuente. Por ejemplo: PHP, Javascript...

3.CICLO DE LA VIDA DEL SOFTWARE

3.1 Ingeniería del software

Disciplina que estudia los principios y metodologías para el desarrollo y mantenimiento de sistemas software.

3.2 Desarrollo del software

Fases principales:

- ANÁLISIS
- DISEÑO
- CODIFICACIÓN
- PRUEBAS
- MANTENIMIENTO

ANÁLISIS

Se determina y define claramente las necesidades del cliente y se especifica los requisitos que debe cumplir el software a desarrollar.
La especificación de requisitos debe:

- Ser completa y sin omisiones
- Ser concisa y sin trivialidades
- Evitar ambigüedades.
- Utilizar lenguaje formal.
- Evitar detalles de diseño o implementación
- Ser entendible por el cliente
- Separar requisitos funcionales y no funcionales(Funcional- Algo que realiza una función, como un botón. No funcional- Lo que no tiene ninguna función, como el fondo, color…)
- Dividir y jerarquizar el modelo (Que haya diferentes categorías)
- Fijar criterios de validación

DISEÑO

Se descompone y organiza el sistema en elementos componentes que pueden ser desarrollados por separado.

Se especifica la interrelación y funcionalidad de los elementos componentes.

Las ac�vidades habituales son:

- Diseño arquitectónico
- Diseño detallado
- Diseño de datos
- Diseño de interfaz de usuario (Lo que el usuario va a ver)

CODIFICACIÓN

Se escribe el código fuente de cada componente.

Pueden utilizarse distintos lenguajes informáticos:

- Lenguajes de programación: C, C++, Java, Javascript, ...
- Lenguajes de otro tipo: HTML, XML, JSON, …

PRUEBAS

El principal objetivo de las pruebas debe ser conseguir que el programa funcione incorrectamente y que se descubran defectos.

Deberemos someter al programa al máximo número de situaciones diferentes.

El primer paso que se realiza en las pruebas es hacer que el programa falle para encontrar los errores antes que los usuarios.

MANTENIMIENTO

Durante la explotación del sistema software es necesario realizar cambios ocasionales.

Para ello hay que rehacer parte del trabajo realizado en las fases previas.

Tipos de mantenimiento:

- Correctivo: se corrigen defectos.
- Perfectivo: se mejora la funcionalidad.
- Evolutivo: se añade funcionalidades nuevas.
- Adaptativo: se adapta a nuevos entornos.

RESULTADO DE CADA FASE

•INGENIERÍA DE SISTEMAS: Especificación del sistema

• ANÁLISIS: Especificación de requisitos del software

• DISEÑO arquitectónico: Documento de arquitectura del software

• DISEÑO detallado: Especificación de módulos y funciones

• CODIFICACIÓN: Código fuente

• PRUEBAS de unidades: Módulos utilizables

• PRUEBAS de integración: Sistema utilizable

• PRUEBAS del sistema: Sistema aceptado

• Documentación: Documentación técnica y de usuario

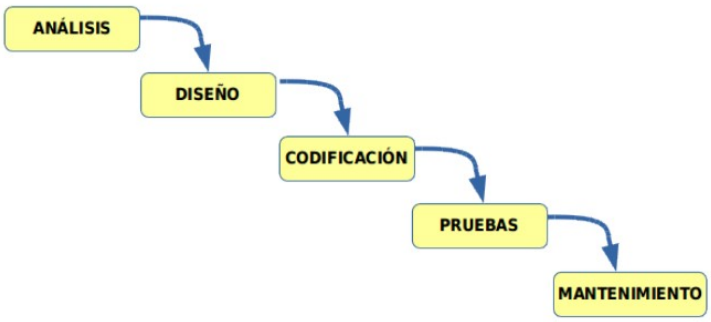
• MANTENIMIENTO: Informes de errores y control de cambios

4.MODELOS DE DESARROLLO

4.1. Modelos de desarrollo de software

- Modelos clásicos (predictivos)
    - o Modelo en cascada
    - o Modelo en V
- Modelo de construcción de prototipos
- Modelos evolutivos o incrementales
    - o Modelo en espiral (iterativos)
    - o Metodologías ágiles (adaptativos)

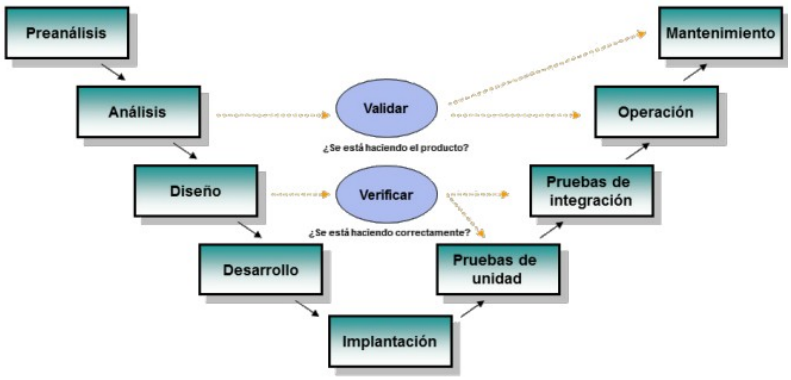
4.2. Modelo en cascada (I)



4.3. Modelo en cascada (II)

- Modelo de mayor antigüedad.
- Identificar las fases principales del desarrollo software
- Las fases han de realizarse en el orden indicado.
- El resultado de una fase es la entrada de la siguiente fase.
- Es un modelo bastante rígido que se adapta mal al cambio contínuo de especificaciones.
- Existen diferentes variantes con mayor o menor cantidad de actividades.

4.4. Modelo en V (I)



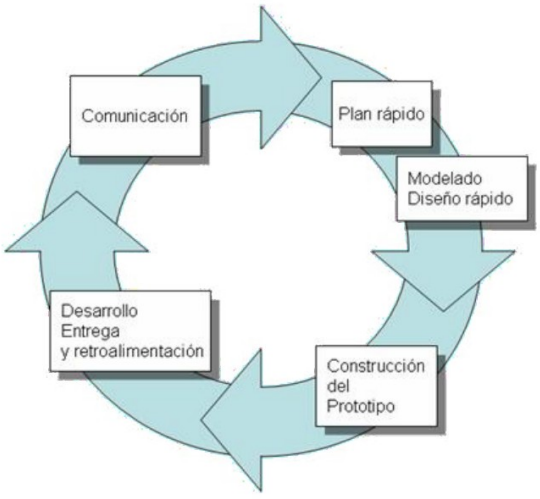
4.5. Modelo en V (II)

• Modelo muy parecido al modelo en cascada.
• Visión jerarquizada con distintos niveles.
• Los niveles superiores indican mayor abstracción.
• Los niveles inferiores indican mayor nivel de detalle.
• El resultado de una fase es la entrada de la siguiente fase.
• Existen diferentes variantes con mayor o menor cantidad de actividades

4.6. Prototipos (I)

“PRUEBA INICIAL”

• A menudo los requisitos no están especificados claramente:
    o Por no existir experiencia previa.
    o Por omisión o falta de concreción del usuario/cliente.



4.8. Prototipos (III)

• Tipos de prototipos:

- o Prototipos rápidos
  - El prototipo puede estar desarrollado usando otro lenguaje y/o herramientas.
  - Finalmente el prototipo se desecha.
- o Prototipos evolutivos
  - El prototipo está diseñado en el mismo lenguaje y herramientas del proyecto.
  - El prototipo se usa como base para desarrollar el proyecto.

.

4.9. Modelo en espiral (I)

• Desarrollado por Boehm en 1988.

• En la actividad de ingeniería corresponde a las fases de los modelos clásicos: análisis, diseño, codificación, ...

Se realiza una vuelta al espiral cada vez que se encuentra en error.

4.10. Modelo en espiral (II)

Aplicado a la programación orientada a objetos

• En la actividad de ingeniería se da gran importancia a la reutilización de código.

4.11. Metodologías ágiles (I)

• Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental.
• Los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.
• El trabajo es realizado mediante la colaboración de equipos auto-organizados y
multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto
plazo.
• Las metodologías más conocidas son:
    o Kanban
    o Scrum
    o XP (eXtreme Programming)

5. Lenguajes de programación

5.1. Obtención de código ejecutable

Para obtener código binario ejecutable tenemos 2 opciones:

- Compilar:

- Interpretar: Cuando compruebo un código, se ejecuta línea a línea.

5.2. Proceso de compilación/interpretación

• La compilación/interpretación del código fuente se lleva a cabo en dos fases:

1. Análisis léxico:Comprueba que las palabras reservadas se han utilizado correctamente.

2. Análisis sintáctico: Analizo que las palabras usadas y compruebo que tengan sentido.

• Si no existen errores, se genera el código objeto correspondiente.
• Un código fuente correctamente escrito no significa que funcione según lo deseado.
• No se realiza un análisis semántico.

5.3. Lenguajes compilados

• Ejemplos: C, C++

• Principal ventaja: Ejecución muy eficiente ya que se le saca el máximo partido al entorno utilizado.(Está más optimizado)

• Principal desventaja: Es necesario compilar cada vez que el código fuente es modificado, se requiere más tiempo.

5.4. Lenguajes interpretados

• Ejemplos: PHP, Javascript

• Principal ventaja: El código fuente se interpreta directamente sin necesitar proceso intermedio.

• Principal desventaja: Ejecución menos eficiente.

5.5. JAVA

Lenguajes compilados e interpretado.

El código fuente Java se compila y se obtiene un código (objeto) binario intermedio denominado bytecode.

Puede considerarse código objeto pero destinado a la máquina virtual de Java en lugar de código objeto nativo.

No está hecho para una máquina concreta.

Después este bytecode se interpreta para ejecutarlo.

5.6. JAVA

Ventajas:

- Estructurado y Orientado a Objetos
- Relativamente fácil de aprender
- Buena documentación y base de usuarios

• Desventajas:

- Menos eficiente que los lenguajes compilados

5.7. Tipos

Según la forma en la que operan:

- Declarativos: indicamos el resultado a obtener sin especificar los pasos.
  - Lógicos: Utilizan reglas. Ej: Prolog
  - Funcionales: Utilizan funciones. Ej: Lisp, Haskell
  - Algebraicos: Utilizan sentencias. Ej: SQL

  Normalmente son lenguajes interpretados.

- Imperativo: indicamos los pasos a seguir para obtener un resultado.
  - Estructurados: C
  - Orientados a objetos: Java
  - Multiparadigma: C++, Javascript

  Los lenguajes orientados a objetos son también lenguajes estructurados.
  Muchos de estos lenguajes son compilados.

• Tipos de lenguajes según nivel de abstracción:

- Bajo nivel: ensamblador
- Medio nivel: C
- Alto nivel: C++, Java (más usado)

5.11. Evolución

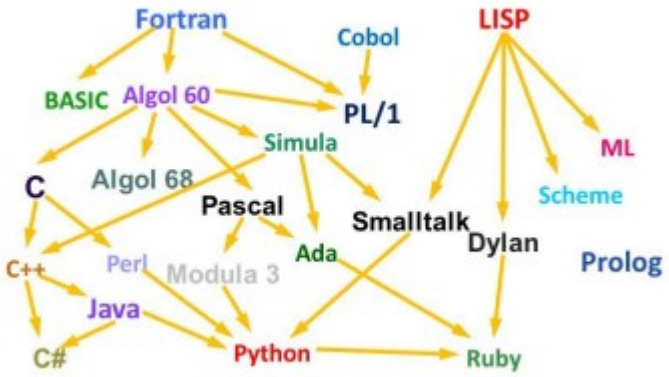
• Código binario

• Ensamblador

• Lenguajes estructurados

• Lenguajes orientados a objetos

5.12. Historia



5.13. Criterios para la selección de un lenguaje

• Campo de aplicación, lo que voy a hacer.
• Experiencia previa, elegir alguno que sepa de qué trata.
• Herramientas de desarrollo, hay que adaptarse a las que tengamos acceso.
• Documentación disponible, algo muy general y lo más popular posible.
• Base de usuarios, cuantos más usuarios lo usen mejor.
• Reusabilidad, aplicación web.
• Portabilidad.
• Imposición del cliente, hacer la aplicación en el entorno que pida el cliente.