UNIDAD 1 - RESUMEN

INTRODUCCIÓN

En esta unidad aprenderemos a:

- Reconocer la relación de los programas con los componentes del sistema informático.
- Diferenciar código fuente, objeto y ejecutable.
- Identificar las fases de desarrollo de una aplicación informática.
- Clasificar los lenguajes de programación.

TIPOS DE SOFTWARE

- De sistema (Sistema operativo, drivers)
- De aplicación (Suite ofimática, Navegador, Edición de imagen, ...)
- De desarrollo (Editores, compiladores, interpretes, ...)

RELACIÓN HARDWARE-SOFTWARE

- Disco duro: almacena de forma permanente los archivos ejecutables y los archivos de datos.
- Memoria RAM: almacena de forma temporal el código binario de los archivos ejecutables y los archivos de datos necesarios.
- CPU: lee y ejecuta instrucciones almacenadas en memoria RAM, así como los datos necesarios.
- E/S: recoge nuevos datos desde la entrada, se muestran los resultados, se leen/guardan a disco, ...

CÓDIGOS FUENTE, OBJETO Y EJECUTABLES

- Código fuente: archivo de texto legible escrito en un lenguaje de programación
- Código objeto: (intermedio): archivo binario no ejecutable.
- Código ejecutable: archivo binario ejecutable.

DESARROLLO DE SOFTWARE

Fases principales:

- Análisis.
- Diseño.
- Codificación.
- Pruebas.
- Documentación.
- 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.
- Evitar detalles de diseño o implementación.
- Ser entendible por el cliente.
- Separar requisitos funcionales y no funcionales.
- Dividir y jerarquizar el modelo.
- 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 actividades habituales son las siguientes:

- Diseño arquitectónico.
- Diseño detallado.
- Diseño de datos.
- Diseño de interfaz.

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.

DOCUMENTACIÓN

El principal objetivo de la documentación es explicar cómo está realizado el código para que otra persona lo entienda y pueda continuarlo.

Se especifica lo que realiza para función del programa.

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

- Análisis: Especificación de requisitos del software.
- Diseño Arquitectónico: Documento de arquitectura del software.
- Diseño Detallado: Especificación de módulos y funciones.
- Codificación: Código fuente.

MODELOS DE DESARROLLO DE SOFTWARE

Modelos clásicos (predictivos):

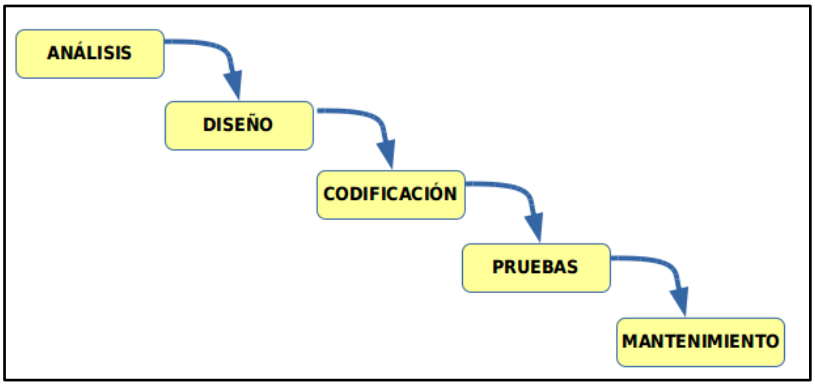
- Modelo en cascada.
- Modelo en V.

Modelo de construcción de prototipos.

Modelos evolutivos o incrementales:

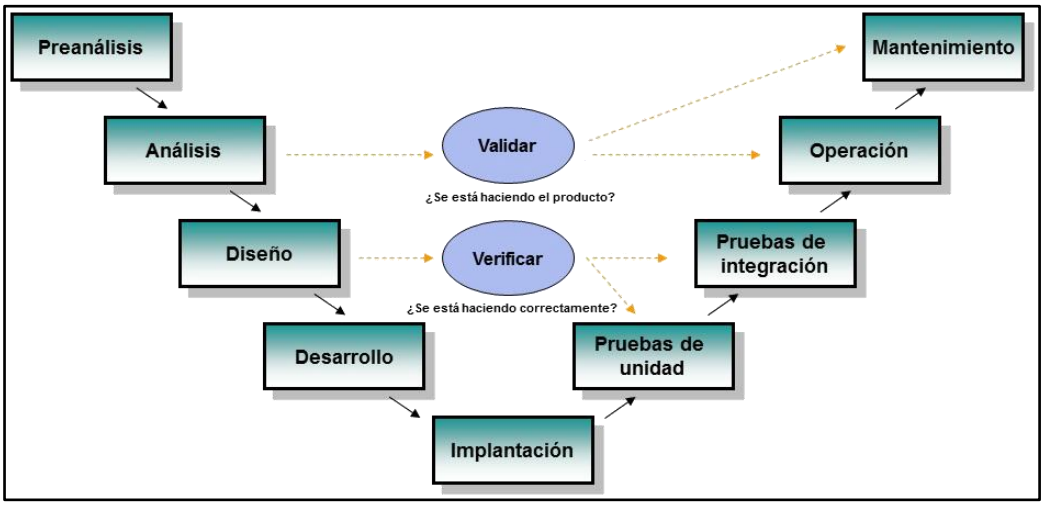
- Modelo en espiral (iterativos).
- Metodologías ágiles (adaptativos).

MODELO EN CASCADA



- Modelo de mayor antigüedad.
- Identifica 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 continuo de especificaciones.
- Existen diferentes variantes con mayor o menor cantidad de actividades.

MODELO EN V



- 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.

PROTOTIPOS

A menudo los requisitos no están especificados claramente:

- Por no existir experiencia previa.
- Por omisión o falta de concreción del usuario/cliente.

Proceso:

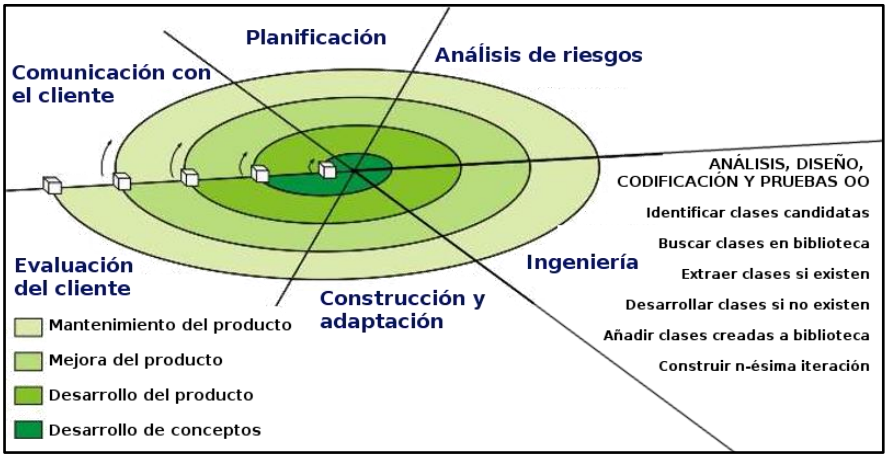
- Se crea un prototipo durante la fase de análisis y es probado por el usuario/cliente para refinar los requisitos del software a desarrollar.
- Se repite el paso anterior las veces necesarias.

Tipos de prototipos:

- Prototipos rápidos:
  - El prototipo puede estar desarrollado usando otro lenguaje y/o herramientas.
  - Finalmente, el prototipo se desecha.

- Prototipos evolutivos:
  - El prototipo está diseñado en el mismo lenguaje y herramientas del proyecto.
  - El prototipo se usa como base para desarrollar el proyecto.

MODELO EN ESPIRAL



- Desarrollado por Boehm en 1988.
- La actividad de ingeniería corresponde a las fases de los modelos clásicos: análisis, diseño, codificación, ...
- Aplicado a la programación orientada a objetos.
- En la actividad de ingeniería se da gran importancia a la reutilización de código.

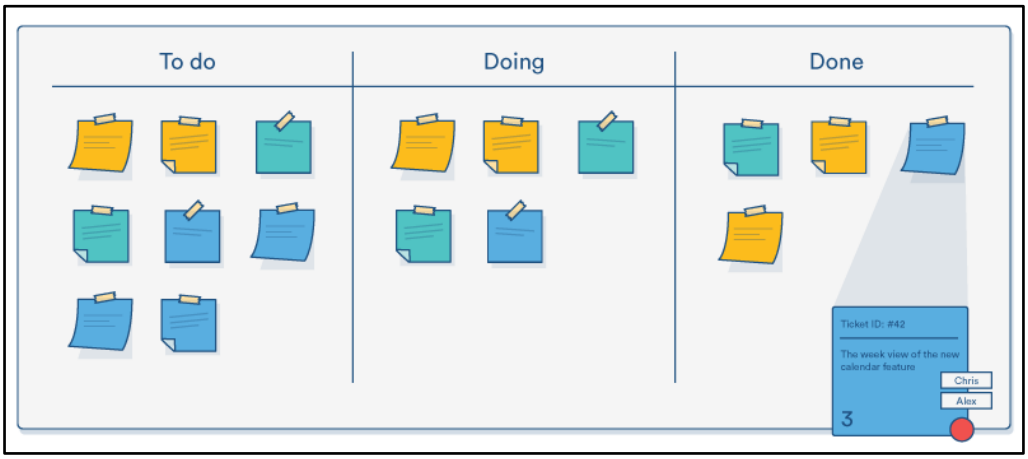
METODOLOGÍAS ÁGILES

- 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:
  - Kanban.
  - Scrum.
  - XP (eXtreme Programming).

Manifiesto por el Desarrollo Ágil

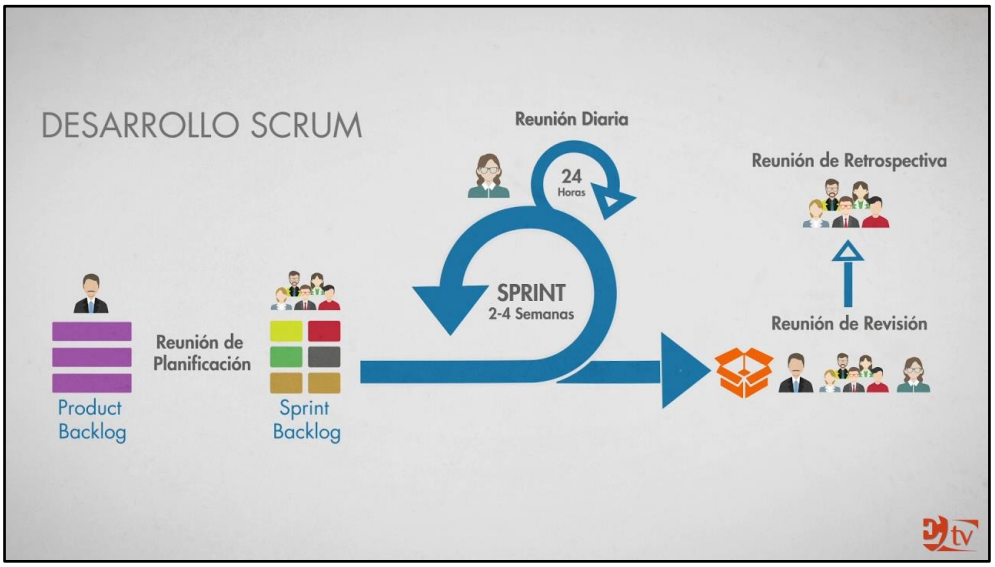
- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

KANBAN



- También se denomina "sistema de tarjetas".
- Desarrollado inicialmente por Toyota para la industria de fabricación de productos.
- Controla por demanda la fabricación de los productos necesarios en la cantidad y tiempo necesarios.
- Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.
- Lean manufacturing.
- Kanban en desarrollo software.

SCRUM



- Modelo de desarrollo incremental.
- Iteraciones (sprint) regulares cada 2 a 4 semanas.
- Al principio de cada iteración se establecen sus objetivos priorizados (sprint backlog).
- Al finalizar cada iteración se obtiene una entrega parcial utilizable por el cliente.
- Existen reuniones diarias para tratar la marcha del sprint.

XP (PROGRAMACIÓN EXTREMA)

Valores:

- Simplicidad.
- Comunicación.
- Retroalimentación.
- Valentía o coraje.
- Respeto o humildad.

Se basa en:

- Diseño sencillo.
- Pequeñas mejoras continuas.
- Pruebas y refactorización.
- Integración continua.
- Programación por parejas.
- El cliente se integra en el equipo de desarrollo.
- Propiedad del código compartida.
- Estándares de codificación.
- 40 horas semanales.

LENGUAJE DE PROGRAMACIÓN

OBTENCIÓN DE CÓDIGO EJECUTABLE

Para obtener código binario ejecutable tenemos 2 opciones:

- Compilar: necesita de un compilador (traductor) que lo traduce y se obtiene un código maquina entendible por la máquina y este es el que se ejecuta.
- Interpretar: es el que necesita de un intérprete, el que lo traduce y ejecuta a la vez.

PROCESO DE COMPILACIÓN/INTERPREACIÓN

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

- Análisis léxico: su función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos
- Análisis sintáctico: comprueba que las sentencias que componen el texto fuente son correctas en el lenguaje.

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.

LENGUAJES COMPILADOS

- Ejemplos: C, C++
- Principal ventaja: Ejecución muy eficiente.
- Principal desventaja: Es necesario compilar cada vez que el código fuente es modificado.

JAVA

- Lenguajes compilado e interpretado.
- El código fuente Java se compila y se obtiene un código binario intermedio denominado bytecode.
- Puede considerarse código objeto, pero destinado a la máquina virtual de Java en lugar de código objeto nativo.
- Después este bytecode se interpreta para ejecutarlo.

Ventajas:

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

Desventajas:

- Menos eficiente que los lenguajes compilados.

TIPOS

Según la forma en la que operan:

- Declarativos: indicamos el resultado a obtener sin especificar los pasos.
- Imperativos: indicamos los pasos a seguir para obtener un resultado.

Tipos de lenguajes declarativos:

- Lógicos: Utilizan reglas. Ej: Prolog.
- Funcionales: Utilizan funciones. Ej: Lisp, Haskell.
- Algebraicos: Utilizan sentencias. Ej: SQL.

Normalmente son lenguajes interpretados.

Tipos de lenguajes imperativos: (Es este tipo de lenguajes, las instrucciones se ejecutan unas tras otras, de manera secuencial)

- 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: (La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan)

- Bajo nivel: ensamblador.
- Medio nivel: C.
- Alto nivel: C++, Java.

Según su forma de ejecución:

- Lenguajes Compilados.
- Lenguajes de Nivel Medio.
- Lenguajes de Bajo Nivel.

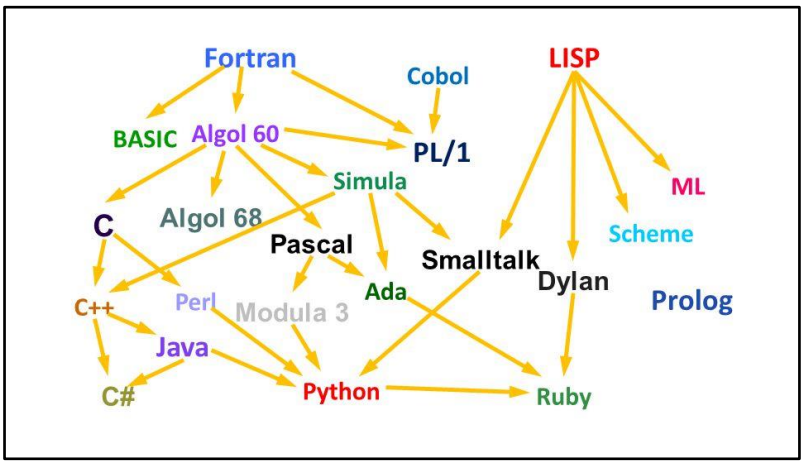
Según su sistema de tipos:

- Lenguajes fuertemente tipados (tipado estático).
- Lenguajes débilmente tipados (tipado dinámico).

EVOLUCIÓN

- Código binario.
- Ensamblador.
- Lenguajes estructurados.
- Lenguajes orientados a objetos.

HISTORIA



CRITERIOS PARA LA SELECCIÓN DE UN LENGUAJE

- Campo de aplicación.
- Experiencia previa.
- Herramientas de desarrollo.
- Documentación disponible.
- Base de usuarios.
- Reusabilidad.
- Portabilidad.
- Imposición del cliente.