1. CONCEPTOS BÁSICOS

Hardware: parte física del ordenador
Software: parte interna del ordenador

Software de Sistema
Software de Aplicación
Software de Desarrollo

→ **Software de Sistema**: Interactúa directamente con el hardware
de bajo nivel, cerca del hardware
- Controla directamente al hardware
SO - Administrador de tarea
firmware: programa (software) de sistema más cercano al hardware. que sirve para controlar programas más simples
Código de programa que se ejecuta en un microcontrolador. Disco duro, reloj inteligente
Buscar qué es un periférico
Drivers: comunica con el firmware/hardware y recibe órdenes del sistema de aplicaciones
Buscar las jerarquías de las memorias RAM, Caché

→ **Software de aplicación**: Más lejos del micro. Los navegadores web, apps que te permiten ver páginas web
- Conjunto de programas que usa el usuario normalmente. Navegador, office, photos

→ **Software de desarrollo**: Programas para programar(para desarrollar nuevos programas)

drivers y controladores de dispositivo son los mismo - entre el sistema operativo y el firmware/hardware

Hardware < drivers/controladores < Sistema Operativo < Sistema de aplicación < Sistema de desarrollo

---

2. Relación Hardware-Software

→ **CPU** (Central Processing Unit) : lee y ejecuta instrucciones almacenadas en memoria RAM
→ **Memoria RAM**: se le llama memoria principal porque es la primera a la que accede el micro. La memoria de almacenamiento permanente (**Disco Duro, SSD,** etc) es la secundaria.
→ **E/S** : Recoge los datos desde la entrada, los muestra y los puede dejar salir desde la salida.
Entrada: Teclado, ratón, etc Salida: Monitor
E/S: USB, Monitor táctil

CPU → Dentro hay registros y memoria caché

Código fuente: código entendible por un ser humano para hacer un programa. Permite modificar el programa de forma sencilla.
Código objeto: Intermedio. Es binario y se genera a partir del código fuente. Es el programa pero no se puede ejecutar
Código ejecutable: Se genera a partir del código objeto. Comienza a ejecutarse en la CPU, son las instrucciones del código fuente, de este al objeto para acabar en el código ejecutable.

Solo sirve para códigos compilados. .exe

El **compilador** lo que hace es interpretar los datos para que el micro lo entienda. Es el que hace que el código funcione en la máquina que sea.

Lenguajes **interpretados** no usan compilador. Estos generan **scripts**(líneas que se ejecutan de manera secuencial). Se suelen usar en servidores, parte administrativa. Python o TypeScript son lenguajes interpretados.
El código fuente mismo es ejecutable.

EL código objeto generado en Java se denomina ByteCode.

draw.io echar un vistazo

3. CICLO DE VIDA DEL SOFTWARE

La palabra ingeniería es sinónimo de diseño. Usas tu INGENIO para buscar una solución a un problema.

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

Desarrollo de Software
Fases:

→ **Análisis**: Analizar los requisitos
→ **Diseño**: Implementar ideas. Donde se define la arquitectura.
→ **Codificación**: Programar el código.
→ **Pruebas**: Probar interacciones con el fin de encontrar fallos.
→ **Mantenimiento**: Con el programa ya documentado, aquí es donde se implementan los parches y actualizaciones.

ANÁLISIS

Análisis: Se determina y define claramente las necesidades del cliente y se especifica los requisitos que debe cumplir el software a desarrollar. Los requisitos no son solo cosa del cliente, sino que de la empresa y el propio equipo de desarrollo.

Especificación de los requisitos

- 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. Que lo pueda entender una persona que no sea técnico.
- Separar requisitos funcionales y no funcionales. NO funcional: *Que sea agradable a la vista* Funcional: Botones, Campos, enlaces, cosas útiles.
- Dividir y jerarquizar el modelo. Dividir en partes el diseño planteado.
- Fijar criterios de validación. Dejar claro los límites que no puede sobrepasar el diseño específico. Lo que le va a valer o no al cliente.

DISEÑO

Aquí el diseñador puede visualizar el producto final.
Se especifican las partes de la aplicación para facilitar su desarrollo.

- **Diseño arquitectónico**: Definir la arquitectura. Definir cómo solucionar el problema a nivel bloque.
- **Diseño detallado**: Definir los problemas individuales de cada bloque.
- **Diseño de datos**: Especificar el tipo de datos usado en la aplicación.
- **Diseño de interfaz de usuario**: Plantear el diseño de cómo interviene con el usuario.

Buscar qué es un mock up: Representación (una imagen estática) de un prototipo de una página web o una aplicación, pero que no es funcional. Está dentro del diseño de interfaz.

CODIFICACIÓN

Escribir el código fuente del programa.

→ **Lenguajes de programación**: C, C++, Java, JavaScript, …
→ **Otros lenguajes**: HTML, XML, …

PRUEBAS

El principal objetivo de las pruebas debe ser conseguir que el programa funcione

incorrectamente y se detecten fallos (bugs).

Se busca detectar el mayor número de fallos posible.
MANTENIMIENTO

Tipo de mantenimiento:

→ **Preventivo**: Se corrigen defectos antes de que se produzcan.
→ **Correctivo**: Corrige defectos
→ **Perfectivo**: Mejora la funcionalidad.
→ **Evolutivo**: Se añade funcionalidades nuevas. (normalmente se paga aparte)
→ **Adaptativo**: Se adapta a nuevos entornos.

Buscar /Imaginar ejemplos de todos.

Modelos de desarrollo

Diseño en cascada

Identifica las fases principales, las fases se realizan en el orden indicado.
Problemas → Lentitud, cuando se está haciendo una fase las demás no hacen nada.
Rígido --> Se adapta mal a los cambios

Modelo en V

Parecido al modelo en cascada pero en el que se puede ver mejor los errores en la fase de análisis como en la de diseño.
Análisis → Operación/Mantenimiento Diseño → Pruebas

Prototipos

Puede ser un mockup, para ver por dónde puede ir el proyecto sin tener que desarrollarlo por completo.

El prototipo se da en la fase de análisis, para perder el mínimo tiempo posible. Se repite el proceso tantas veces como sea necesario.

Dos tipos: Prototipos rápidos y prototipos evolutivos.

Modelo en espiral

Centrado en minimizar los riesgos. En contacto constante con el cliente
Aplicado a programación tiene la idea de darle importancia a la reutilización del código.

Metodologías ágiles

Metodologías que se adaptan más al cliente
El trabajo se realiza mediante la colaboración del equipo de desarollo y el equipo del cliente, con mucha interacción entre estos
Equipo multidisciplinario: Son autónomos, tienen variedad de personal como para a llevar el proyecto entero si hiciera falta.
Desarrollo iterativo, poco a poco

Metodologías más conocidas:
Kanban, Scrum

Son ágiles porque se le da feedback al cliente muy a menudo.

Kanban

Sistema de **tarjetas**.
Puntaje: relación entre la complejidad de la tarea y el tiempo que se tarda en realizarla
Historia: Un requisito especificado. Es la tarea final (crear un botón)
Microtareas: Todos los pasos que hay que dar para realizar la historia (procesos y tareas que implican crear y añadir un botón)

Scrum

Hay unas tareas a realizar, y tiene que quedar muy claro quién las hace y cómo se deben hacer.
El trabajo se realiza por **sprints**.
Las iteraciones se llaman sprints y duran de 2 a 4 semanas.
Sprint backlog. Backlog de todas las historias para desarrollarlas durante el sprint.
Al final de cada sprint el programa tiene que tener alguna mejora con respecto al sprint anterior.
Incremento estado del producto al final de cada sprint

Al final de cada sprint también hay una reunión del equipo de desarrollo para sincronizar las actividades y crear un plan de trabajo. Se le explica al cliente de las tarea plabeadas lo que se ha hecho y lo que no durante el sprint. Cada 2-4 semanas.
También los hay diarios, (llamada daily) para los planes a corto plazo.
Backlog: Lista de tareas por hacer
Scrum review: reunión para evaluar el incremento del proyecto
Srum retrospective: Reunión de feedback para donde se evalúa la correcta aplicación del scrum
Product Owner: La "Voz del Cliente". Hace de enlace entre el cliente y el equipo de desarrollo. Tiene que conocer la arquiteectura del programa etc, puede ser un desarrollador senior por ejemplo.
Scrum Master: El que se asegura que se sigue la metodología Scrum.
Desventaja: La implementación no es intrínseca de la metodología, a pesar de ser ágil,a veces podemos quedar esclavos del método.

Retrofit: Una prueba que se realiza para ver si algo es defectuoso.

Programación Extrema

Diseño sencillo
Tareas pequeñas
Se suele trabajar por parejas
El cliente está integrado en el equipo de desarrollo.

4. Lenguajes de Programación

Se busca tener un código **ejectuable**:
- -- Compilado
- -- Interpretado

Proceso de compilado/interpretación --> 2 Fases
- -- Análisis **léxico**
- -- Análisis **sintáctico**

Compilado --> C, C++
- **Ventaja**: Ejecución muy eficiente
- **Desventaja**: Hay que compilar todo el código cada vez que se modifica

Interpretado --> PHP, JavaScript, Python
- (**No** se genera código objeto)
- **Ventaja**: Se interpreta directamente
- **Desventaja**: Ejecución menos eficiente

Java --> Compilado **E** interpretado

Se compila --> Genera codigo binario intermedio --> Se llama Bytecode

El bytecode no es concreto para un solo procesador

No es tan eficiente porque se limita a la máquina virtual de Java y no a un procesador concreto

Ventajas:

Lenguaje Orientado a Objetos

Clases --> Métodos --> Operaciones que se ejecutan

Se controla bien la documentación y los usuarios, partes privadas y públicas, etc

Desventaja: Al ser interpretado, no es del todo eficiente

- -

Volvemos a los lenguajes en general

Tipos:

- **Declarativos**:
  - Lógicos: Reglas
  - Funcionales: Funciones
  - Algebraicos: Sentencias
- **Imperativos**:
  - Estructurados: C
  - Orientados a objetos: Java
  - Multiparadigma: C++, JavaScript

Según nivel de **abstracción**: A más nivel tenga, más eficiente es el código. Pero pierde un poco de especificación, es más general.

- **Bajo** nivel: Emsamblador
- **Medio** Nivel: C
- **Alto** nivel: C++, Java

Evolución:

Del más bajo nivel hasta que se ha llegado al más alto nivel

| Código Binario
| Ensamblador
| Lenguajes Estructurados
V Lenguajes Orientados a Objetos

Criterios para la **selección** de un lenguaje:

- Campo de aplicación
- Experiencia previa
- Herramientas de desarrollo
- Documentación disponible

- Base de usuarios
- Reusabilidad
- Transportabilidad
- Imposición del cliente

- -

Repaso general de la unidad

RAM: también se trae datos del programa para almacenarlos (cuando un programa por segunda vez se ejecuta antes)

Código fuente > (Código objeto)* > Código ejecutable
* Sólo si el lenguaje es compilado

Java va aparte, tiene su ByteCode, solo ejecutable en la máquina virtual de Java

Análisis > Diseño > Codificación > Pruebas > Mantenimiento

Modelos de desarrollo:

Cascada, V, Prototipos, Espiral -- Rígidos
Kanban, Scrum, XP -- Ágiles

Scrum:
Sprint backlog - De todas las tareas que había en la pizarra, las seleccionadas para hacer en un sprint concreto
Un entregable no puede ser una versión de código que no ejecutable o que no funciona correctamente
Un equipo de Scrum no son solo desarrolladores, hay variedad de funciones
En el sprint planning se selecciona el sprint backlog
Scrum Review - Reunión donde se evaúa como ha ido el sprint
Scrum Retrospective - Reunión donde se evalúa cómo se ha utilizado el Scrum dentro del Sprint

Análisis léxico - Analiza las palabras utilizadas
Análisis sintáctico - Revisa que la organización y el orden de las palabras es el correcto.