CONCEPTOS GENERALES

María Oliver Casanova

Tabla de conetnidos

1.1. Conceptos generales

La ingeniería del software es, básicamente, la forma ordenada y con cabeza de crear programas o aplicaciones. No se trata solo de escribir código: se trata de planificar, diseñar, probar, mantener y mejorar el software como si fuera un proyecto de construcción... pero en digital.

Imagínate que vas a construir una casa: antes de poner un ladrillo, haces planos, eliges materiales, calculas costes, y piensas en la instalación eléctrica y el agua. Pues con una web o una app pasa igual: hay que planificar la estructura, el diseño, las funcionalidades, la seguridad, etc.

En el mundo web, la ingeniería del software incluye todo el ciclo de vida de una página o aplicación: desde pensar la idea hasta subirla al servidor y mantenerla en el tiempo.

Ejemplo real:

si una ONG quiere una web de voluntariado, hay que definir qué podrá hacer la gente (registrarse, apuntarse a actividades, donar, etc.), cómo se mostrará la información y qué herramientas se usarán (WordPress, React, bases de datos, etc.).

Idea

Planificar

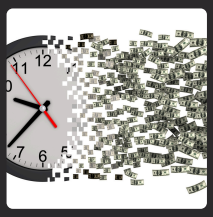
Diseñar

Probar

1.2. Objetivos

Estandarizar

Ahorrar tiempo y dinero



Usuario final

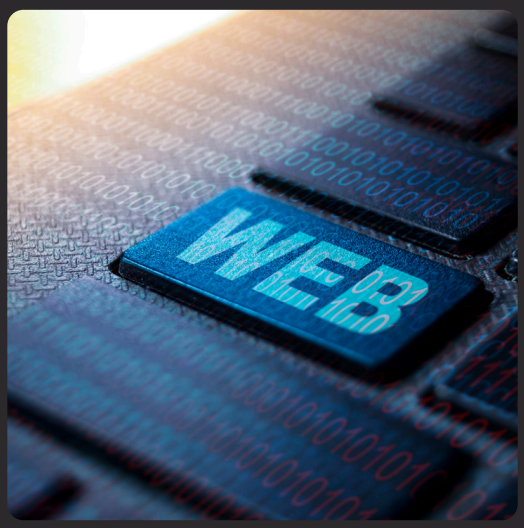
El objetivo principal es crear software de calidad sin morir en el intento.
Para eso, se busca:

- Estandarizar los procesos: que el desarrollo no dependa de cómo se levante el programador ese día. Que haya una forma clara y repetible de hacer las cosas.
- Ahorrar tiempo y dinero: planificando bien desde el principio se evitan los típicos "¡esto no era lo que quería el cliente!".
- Aumentar la calidad: aplicando pruebas, revisiones y control de versiones para que el producto final no tenga errores ni sea un caos.
- Fomentar el trabajo en equipo: unir a programadores, diseñadores y gestores en una misma visión.
- Pensar siempre en el usuario final: que la web sea accesible, rápida y fácil de usar para todo el mundo (siguiendo normas como las WCAG 2.2).

Ejemplo práctico:

Si haces una web para una escuela de idiomas y nadie puede encontrar el botón de "matrícula", o tarda 10 segundos en cargar, algo ha fallado en la ingeniería del software, aunque el diseño sea precioso.

1.3. Características



El software no es como una silla o una casa. No se toca, se crea, se ejecuta y se mejora. Tiene unas peculiaridades:

- Intangible: no se fabrica, se desarrolla. No hay piezas físicas, todo está en código.
- Evolutivo: nunca está "terminado del todo". Siempre hay algo que actualizar o mejorar.
- Personalizable: se adapta a distintos usuarios o contextos. Un ejemplo claro: la misma web puede verse diferente en móvil y en ordenador.
- Dependiente del entorno tecnológico: cambia según el navegador, el servidor, o el lenguaje que se use.

Ejemplo:

Una web que funciona genial en Chrome puede dar fallos en Safari si no se ha probado bien. Por eso el software depende tanto del entorno.

1.4. Principios básicos

Modularidad

Dividir el proyecto en partes pequeñas (módulos) que se puedan trabajar por separado.

➔ **Ejemplo:** en una tienda online, separar el módulo de "productos", el de "usuarios" y el de "pagos".

Documentación

dejar apuntado todo lo importante (cómo se instala, cómo se usa, cómo se actualiza).

➔ **Ejemplo:** el típico README.md de GitHub.

Verificación y validación

comprobar que el software funciona y que hace lo que el cliente quiere

Abstracción

ocultar los detalles internos y dejar claro lo que cada parte hace.

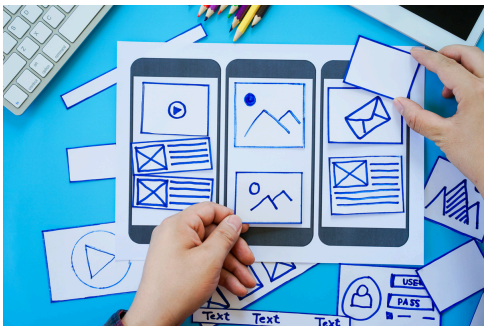
➔ **Ejemplo:** usas una API sin saber cómo está programada por dentro, solo lo que te devuelve.

Reutilización

no reinventar la rueda. Si ya existe una librería o un plugin que hace algo, ¡úsalo!

Control de versiones

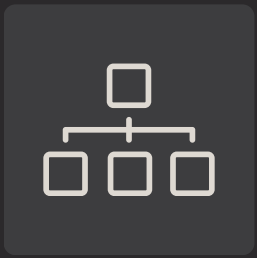
para no perder cambios y poder volver atrás si algo sale mal.



Mantenimiento evolutivo

actualizar y mejorar con el tiempo sin romper lo que ya funciona.

1.5. Paradigmas de desarrollo de software



Estructurado #1

el clásico, con pasos bien definidos (como en C o Pascal).

Orientado a objetos #2

organiza el código en “clases” y “objetos” (muy común en Java, PHP o Pyth

Orientado a componentes/servicios #3

se divide en piezas independientes que se comunican entre sí (microservicios, APIs).

Orientado a eventos #4

reacciona a acciones del usuario (clicks, movimientos del ratón, etc.), típico en JavaScript.

Orientado a datos #5

reacciona a acciones del usuario (clicks, movimientos del ratón, etc.), típico en JavaScript.

UX-driven #6

el diseño y la experiencia de usuario guían todo el proceso.

Ejemplo práctico:

Una web hecha en Wordpress funciona bajo el paradigma de elementos. Cada botón, menú o formulario es una pieza independiente que se puede reutilizar en varias partes del sitio.

1.6. Buenas prácticas en desarrollo web

Un ingeniero o ingeniera web con experiencia suele seguir estas rutinas:

- Usar arquitecturas limpias: separar lo visual, lo lógico y lo funcional
- Documentar todo: usar Notion, Wikis o README para dejar constancia de decisiones y procesos.
- Automatizar tareas: usar herramientas que limpien código, compriman archivos o desplieguen la web automáticamente
- Hacer testing: probar que cada parte funciona antes de publicar
- Revisar: otras personas revisan la web para detectar fallos.
- Gestionar proyectos con metodologías ágiles: usar tableros Kanban o Scrum, con tareas cortas y entregas frecuentes.
- Entrega continua: subir nuevas versiones sin parar el servicio.
- Pensar siempre en el usuario: accesibilidad, velocidad, compatibilidad móvil y buen diseño.

Caso práctico:

Un equipo desarrolla una web para una fundación. Cada semana entregan una parte (blog, formularios, donaciones) y hacen pruebas en dispositivos reales. Si algo falla, lo corrigen antes de la siguiente entrega. Esa es una buena práctica de ingeniería web.

1.7. Estándares y marcos de referencia

ISO/IEC 12207

explica cómo debe organizarse todo el ciclo de vida del software

ISO/IEC 25010

define qué significa que un software sea “de calidad”

OWASP Top 10

lista de los errores de seguridad más comunes (por ejemplo, inyecciones SQL o contraseñas débiles)

IEEE 830

dejar apuntado todo lo importante (cómo se instala, cómo se usa, cómo se actualiza)

WCAG 2.2

reglas para que las webs sean accesibles a todas las personas

Ejemplo:

Si tu web tiene texto con poco contraste o botones muy pequeños, no cumple la WCAG 2.2. Si además guarda contraseñas sin cifrar, está rompiendo todas las normas OWASP

1.8. Rol del ingeniero/a de software web

El ingeniero o ingeniera de software web no solo programa: organiza, coordina y mejora todo el proceso de desarrollo.
Es como un/a director/a de orquesta digital .
Sus tareas incluyen:

- Planificar tiempos, costes y prioridades.
- Evaluar riesgos y buscar soluciones antes de que haya problemas.
- Promover herramientas modernas (GitHub, CI/CD, entornos cloud).
- Facilitar la comunicación entre programadores, diseñadores y clientes.
- Asegurar que el resultado final sea seguro, accesible, rápido y bonito.

Caso real:

María, ingeniera web, coordina a tres personas para crear una web de comercio justo. Mientras una diseña el interfaz en Figma, otro programa en React y otro monta el servidor en Plesk, ella se encarga de revisar que todo funcione, esté documentado y cumpla plazos. Esa es la esencia del rol.