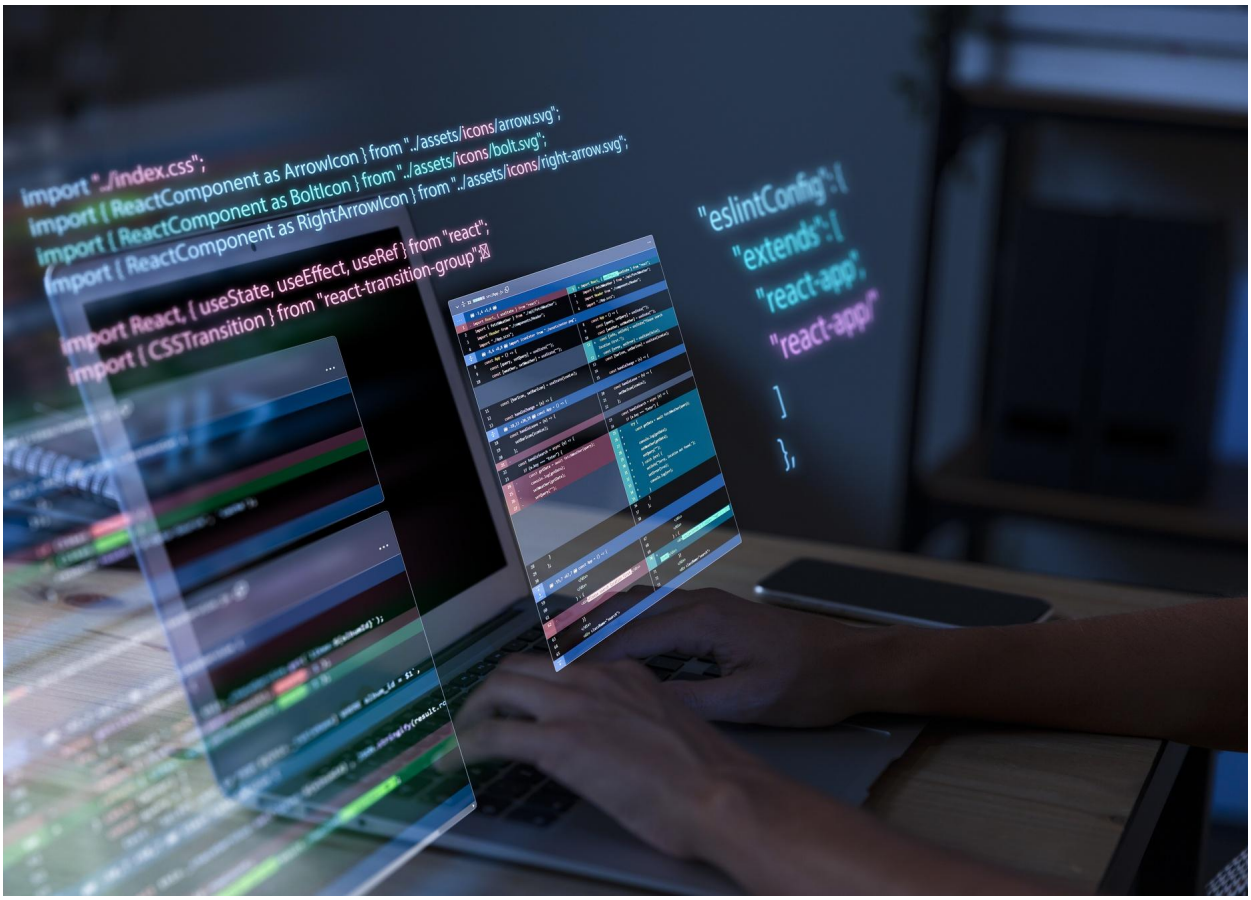
1º DAM & DAW /ENTORNOS DE DESARROLLO

UNIDAD 1

Introducción al desarrollo de software y lenguajes



Autor: Luz María Álvarez Moreno

Fecha: 30/9/2025

Sesión 1: ¿Qué es el software y cómo se crea?

Teoría y Definiciones

- **Hardware**
    - Parte física del ordenador: CPU, memoria, disco, periféricos.
    - Ejemplo: teclado, monitor, impresora.
    - El “cuerpo” de la máquina.
- **Software**
    - Programas, instrucciones y datos que permiten que el hardware funcione.
    - Intangible: no se toca, se ejecuta.
    - La “mente” que da órdenes al hardware.
- **Relación Hardware – Software**
    - Sin software: un ordenador es inútil.
    - Con software: se transforma en calculadora, procesador de texto, navegador, etc.

Tipos de Software

1. **De sistema**: base del funcionamiento:
    - SO: Windows, Linux, macOS.
    - Drivers: impresora, tarjeta gráfica.

- Utilidades básicas: antivirus, gestores de archivos.
- Ejemplo: Windows gestiona CPU + impresora.

2. **De aplicación**: usado directamente por el usuario:
   - Procesadores de texto, hojas de cálculo, navegadores, apps móviles.
   - Ejemplo: Word, Spotify, Zoom.
3. **De desarrollo**: usado por programadores:
   - IDEs (Eclipse, IntelliJ, VS Code).
   - Compiladores, frameworks, SDKs.
   - Ejemplo: Android Studio para apps móviles.

Proceso de Creación de Software (Ciclo de Vida Básico)

1. **Análisis de requisitos**: identificar necesidades del usuario.
2. **Diseño**: arquitectura, UI/UX, estructura de datos.
3. **Programación**: convertir diseño en código (Java, Python…).
4. **Pruebas**: unitarias, de integración, de aceptación.
5. **Mantenimiento**: correcciones, parches, mejoras.

Idea clave: programar es solo una parte.

- Crear software = visión + organización + calidad.

Ejemplo Cotidiano: Tu móvil

- **SO:** Android o iOS gestionando memoria y batería.
- **Apps:** WhatsApp, Spotify, Instagram.
- **Software de desarrollo:** SDKs y frameworks que usaron los programadores.

Caso de Estudio: WhatsApp

- **Contexto:** nació en 2009 como alternativa gratuita a SMS.
- **Estrategia:**
    - Interfaz simple.
    - Backend en Erlang (soporta millones de mensajes).
    - Evolución continua: llamadas, video, cifrado, Business.
- **Resultado:** +2.000M usuarios en 180 países.

Clave: simplicidad + evolución iterativa.

Herramientas y Consejos

- Diferencia tipos de software en la práctica:
    - Instala Linux → sistema.
    - Usa Google Docs → aplicación.
    - Descarga VS Code → desarrollo.
- Empieza con proyectos pequeños:
    - Calculadora en Python.

    - Blog con HTML, CSS y JS.

- Usa metodologías modernas: Scrum, Kanban.

Preguntas de Reflexión

- ¿Qué pasaría si existiera hardware sin software?
- ¿Qué software usas a diario y cómo se clasificaría?
- ¿Qué es más importante: un buen análisis o una buena programación?

Mitos y Realidades

- **Mito**: *"El software se crea solo programando"*.
  Realidad: Programación es solo una fase; sin análisis y pruebas el software falla.
- **Mito**: *"Si funciona la primera versión, ya está"*.
  Realidad: Todo software requiere mantenimiento y actualizaciones.

Sesión 2: Lenguajes de programación y clasificación

1. Definición

- Lenguaje de programación = conjunto de reglas para dar instrucciones al ordenador.
    - Puente entre pensamiento humano y máquina.

2. Clasificación

1. **Según nivel de abstracción:**

- **Bajo nivel:** ensamblador.
    - Preciso, difícil, lento de programar.
- **Alto nivel:** Python, Java.
    - Más cercano al humano, productivo.

2. **Según paradigma:**
    - **Imperativo**: paso a paso (C).
    - **Declarativo**: qué quiero, no cómo (SQL, HTML).
    - **Funcional**: matemático, funciones puras (Haskell, Scala).
    - **Orientado a objetos:** clases y objetos (Java, C++, C#).
3. **Según ámbito de aplicación:**
    - **Propósito general:** Java, Python, C++.
    - **Específicos de dominio** (DSL): SQL (BD), R (estadística), MATLAB (cálculos).

Realidad Profesional

- Proyectos actuales usan varios lenguajes a la vez.
- Ejemplo web moderna:
    - Frontend: HTML, CSS, JS.
    - Backend: Python/Java/PHP.
    - DB: SQL.

Caso de Estudio: Coursera

- **Frontend:** JS (React).
- **Backend:** Python (Django).
- **BD:** PostgreSQL.
- **Analítica:** R y Python.
- **Infraestructura:** AWS, Kubernetes.
- **Resultado:** escalabilidad global, +100M usuarios.

Herramientas y Consejos

- Aprende un lenguaje multipropósito: Python, Java o JS.
- Complementa con SQL o R.
- Practica online en Replit, HackerRank, LeetCode.
- Revisa rankings (TIOBE, Stack Overflow Survey).

Preguntas de Reflexión

- ¿Qué lenguaje elegirías para hacer una app móvil? ¿Y para análisis de datos?
- ¿Por qué en proyectos reales se usan varios lenguajes?

7. Mitos y Realidades

- **Mito**: *"Con saber un lenguaje es suficiente"*.

  Realidad: Hoy se necesitan varios.

- **Mito**: *"Aprender un lenguaje sirve para toda la vida"*.

  Realidad: Los lenguajes evolucionan; hay que actualizarse.

Sesión 3: Proceso de ejecución (compilación, interpretación y bytecode)

Modelos de ejecución

1. **Compilación:**
    - Traductor genera ejecutable completo.
    - Ej.: C, C++.
    - Rápido en ejecución.
    - Lento al modificar (hay que recompilar).
2. **Interpretación:**
    - Se ejecuta línea a línea.
    - Ej.: Python, JS.
    - Flexible, ideal para prototipos.
    - Más lento.
3. **Bytecode + Máquina Virtual:**
    - Genera código intermedio.
    - Ej.: Java (JVM), C# (CLR).
    - Portabilidad.

- Algo más lento (aunque con JIT mejora mucho).

Característica	Lenguajes de Alto Nivel	Lenguajes de Bajo Nivel
Cercanía al hardware	Lejanos al hardware, más cercanos al lenguaje humano.	Muy cercanos al hardware y a la arquitectura del procesador.
Sintaxis	Fácil de leer y entender (ej. if, while, print).	Compleja, basada en instrucciones máquina o ensamblador.
Ejemplos	Java, Python, C#, JavaScript, C++, Ruby.	Ensamblador (Assembly), lenguaje máquina (binario).
Portabilidad	Alta: el mismo código puede ejecutarse en distintos sistemas con pocas modificaciones.	Baja: depende del tipo de procesador o arquitectura.
Velocidad de ejecución	Menor, ya que requiere interpretación o compilación adicional.	Muy alta, ya que las instrucciones se ejecutan directamente por la CPU.
Facilidad de programación	Alta: incluyen abstracciones, estructuras de control y manejo automático de memoria.	Baja: el programador debe gestionar manualmente recursos y direcciones de memoria.

Uso principal	Aplicaciones, software, inteligencia artificial, desarrollo web, etc.	Controladores, sistemas embebidos, firmware, optimización de hardware.

Ejemplo Actual

- Python: interpretado, pero genera bytecode .pyc.
- JS: interpretado + JIT.

Caso de Estudio: Banca en Java

- Necesidad: seguridad + portabilidad.
- Evaluaron C++ (rápido pero no portable) y Python (portable pero lento).
- Solución: Java (bytecode en JVM).
- Resultado: mismo código en múltiples países + seguridad robusta.

4. Herramientas y Consejos

- Haz un "Hola Mundo" en C++, Python y Java para ver diferencias.
- Usa JVM y CLR para probar portabilidad.
- Experimenta con profiling (VisualVM, PySpy).

5. Preguntas de Reflexión

- ¿Qué modelo usarías para un videojuego 3D? ¿Y para un prototipo de IA?
- ¿Por qué es clave la portabilidad en software bancario?

6. Mitos y Realidades

- **Mito**: *"Interpretados siempre son lentos"*.

  Realidad: Con JIT pueden acercarse a compilados.

- **Mito**: *"Si compila, está perfecto"*.

  Realidad: Puede tener errores lógicos, bugs o vulnerabilidades.

Sesión 4: Ciclo de vida y metodologías de desarrollo

1. Fases comunes

1. Análisis de requisitos.
2. Diseño.
3. Implementación.
4. Pruebas.
5. Despliegue.
6. Mantenimiento.

Modelos de organización

- **Cascada:**
  - Secuencial, rígido.
  - Claridad y control.
  - Poco flexible.

    - Ejemplo: sistemas médicos, aeroespaciales.

- **Iterativos/Ágiles:**
    - Ciclos cortos (Scrum, Kanban).
    - Flexibilidad, feedback continuo.
    - Requiere disciplina y cliente involucrado.
    - Ejemplo: startups, apps móviles.

Caso de Estudio: App de reservas

- **Cascada:** producto tardío y obsoleto.
- **Ágil:** MVP rápido + mejoras → éxito competitivo.

Herramientas y Consejos

- Cascada: Microsoft Project.
- Ágil: Jira, Trello, ClickUp, Asana.
- Documentación: Notion, Confluence, Figma.
- Calidad: pruebas automatizadas, CI/CD.

Preguntas de Reflexión

- ¿Qué metodología usarías en un hospital? ¿Y en una startup?
- ¿Qué ventajas tiene entregar un MVP rápido?

Mitos y Realidades

- **Mito:** *"Ágil = improvisación"*.

- **Realidad:** Ágil planifica, pero en ciclos cortos y flexibles.

Comparativa: Modelo en Cascada vs. Metodologías Ágiles

Aspecto	**Modelo en Cascada**	**Metodologías Ágiles (Scrum, Kanban, etc.)**
Estructura del proceso	Secuencial y lineal (cada fase depende de la anterior).	Iterativa e incremental (ciclos cortos con entregas continuas).
Orden de fases	1 Análisis → 2 Diseño → 3 Implementación → 4 Pruebas → 5 Despliegue → 6 Mantenimiento.	Pequeños ciclos que incluyen todas las fases (análisis, diseño, desarrollo y pruebas en cada sprint).
Flexibilidad	**Muy baja**: los cambios después del diseño son costosos.	**Muy alta:** permite ajustes frecuentes según el feedback del cliente.
Entrega del producto	Se entrega el producto completo al final del proceso.	Se entregan versiones funcionales (MVP) en intervalos cortos.
Requisitos del cliente	Se definen al inicio y permanecen fijos.	Pueden evolucionar durante el desarrollo.

Comunicación con el cliente	**Escasa**: principalmente al principio y al final.	**Constante**: reuniones periódicas, revisiones y feedback continuo.
Documentación	Muy detallada y formal en cada fase.	Ligera y dinámica, enfocada en la colaboración y la comunicación directa.
Gestión del proyecto	Basada en planificación rígida y control estricto.	Basada en adaptabilidad, autoorganización y trabajo en equipo.
Control de calidad	Se realiza al final del desarrollo (fase de pruebas).	Se integra en todo el proceso (pruebas continuas, integración continua).
Rol del equipo	**Jerárquico**: analistas, diseñadores y programadores trabajan por separado.	**Colaborativo**: equipos multidisciplinares que se autoorganizan.
Herramientas típicas	Microsoft Project, Gantt, diagramas de flujo.	Jira, Trello, Asana, ClickUp, Figma.
Ventajas principales	Claridad en la planificación, control y documentación exhaustiva.	Flexibilidad, rapidez en la entrega, mejora continua y satisfacción del cliente.

Desventajas principales	Poca adaptabilidad, alto riesgo de producto obsoleto, baja comunicación.	Requiere disciplina, compromiso del cliente y gestión constante.
Ejemplos de uso	Proyectos con requisitos muy definidos y poco cambiantes (sistemas médicos, aeroespaciales, banca).	Proyectos innovadores o cambiantes (startups, apps móviles, desarrollo web).

Resumen

- **Software = mente del ordenador, indispensable para el hardware.**
- **Clasificación:** sistema, aplicación, desarrollo.
- **Lenguajes:** bajo/alto nivel, paradigmas, aplicaciones.
- **Ejecución:** compilado, interpretado, bytecode.
- **Ciclo de vida:** fases y metodologías (cascada vs ágil).
- **Ejemplos reales:** WhatsApp, Coursera, banca, startups.
- **Mitos desmentidos:** no basta programar, no basta un lenguaje, no basta compilar.

Ejercicio práctico: "Del lenguaje de la máquina al lenguaje humano"

Objetivos

- Comprender la diferencia entre un lenguaje **de bajo nivel** y uno **de alto nivel**.
- Observar cómo cambia la **forma de escribir**, **leer** y **entender** un mismo programa.
- Ejecutar ejemplos reales con ayuda del profesor, **sin escribir código desde cero**.
- Familiarizarse con los entornos de desarrollo **NetBeans** y **Visual Studio Code**.

Parte 1 — Introducción: ¿Qué es cada nivel?

Bajo nivel:

Lenguajes que hablan casi directamente con el procesador (como el ensamblador).

- Muy detallados, muy rápidos, pero difíciles de entender.

Alto nivel:

Lenguajes que usan palabras parecidas al lenguaje humano (como Java o Python).

- Más lentos, pero más fáciles de leer y escribir.

Parte 2 — Ejemplos práctico paso a paso .zip con códigos "Hola Mundo"en diferentes lenguajes de Programación.

1. ¿Qué diferencias observas entre el código de C y el de Python?
2. ¿Cuál te parece más fácil de leer?
3. ¿Qué lenguaje crees que usa un robot o un microcontrolador?
4. ¿Cuál sería más apropiado para hacer una página web?
5. ¿Por qué crees que existen tantos lenguajes distintos?

Ejemplos Metodologías.

Modelo en Cascada – Sistema de Control Hospitalario

Proyecto:

Desarrollo de un **sistema de gestión hospitalaria** que administre:

- Historias clínicas de los pacientes.
- Inventario de medicamentos.
- Turnos del personal médico y de enfermería.

Se trata de un proyecto grande, complejo y sensible, donde un fallo puede tener consecuencias graves.

Características del proyecto

- Requisitos **claros y definidos desde el principio**
    - Antes de empezar, el hospital define exactamente qué necesita.
      No se permiten improvisaciones, ya que los procesos médicos están regulados por leyes y protocolos.
- Cambios **poco frecuentes**
    - Una vez aprobado el diseño, modificar algo es difícil, porque afectaría a muchos módulos del sistema (pacientes, medicación, seguridad, permisos…).

- Altos estándares de **seguridad, fiabilidad y validación legal**.
    - El sistema gestiona datos médicos personales, por lo que debe cumplir normas legales (como protección de datos y auditorías sanitarias).

Por estas razones, el proyecto necesita **control, trazabilidad y documentación completa**, no rapidez ni improvisación.

Aplicación del modelo en cascada

1. **Análisis de requisitos:** se documentan de forma exhaustiva las necesidades del hospital.
    - Los analistas se reúnen con médicos, enfermeros y personal administrativo.
    - Se definen **todas las funciones** del sistema:
        - registro de pacientes
        - consultas
        - recetas
        - turnos
        - inventarios
        - etc.
    - Se documenta todo en un **documento de especificaciones funcionales**, aprobado por la dirección del hospital.

2. **Diseño:** se definen todas las pantallas, bases de datos y reglas del sistema.
    - Se crean los
        - **diagramas de flujo**
        - **bases de datos**
        - **pantallas**
        - **reglas de negocio**.
    - Se definen los **niveles de acceso y permisos** (médico, enfermero, administrativo).
    - Todo el diseño se revisa y aprueba antes de empezar a programar.
    - El diseño garantiza que el sistema sea **seguro, coherente y escalable**, y que cumpla la normativa sanitaria.

3. **Implementación:**
    - Los programadores traducen el diseño en código.
    - Se crean los módulos del sistema (pacientes, medicamentos, horarios, informes, etc.).
    - No se hacen cambios de requisitos: se sigue el plan exactamente como fue diseñado.
    - En proyectos críticos, los cambios improvisados pueden romper la integridad del sistema o provocar errores graves.

4. **Pruebas:** se validan los módulos y la seguridad antes del despliegue.
    - Se revisa cada módulo de forma aislada (pruebas unitarias) y después el sistema completo (pruebas integradas).
    - Se hacen pruebas de **seguridad**, **rendimiento** y **cumplimiento normativo**.
    - No se puede arriesgar a que un error cause pérdida de datos o errores médicos.
    - Solo cuando todo funciona al 100% se pasa al despliegue.
5. **Despliegue:** el sistema se instala y los usuarios reciben formación.
    - Se instala el sistema en los equipos del hospital.
    - Se entrena al personal para usarlo correctamente.
    - Se redactan manuales de usuario y guías de emergencia.
    - En entornos hospitalarios, el personal debe estar bien formado para evitar errores de uso o introducción de datos.
6. **Mantenimiento:** corrección de errores y actualizaciones puntuales.
    - Se corrigen errores detectados después del uso real.
    - Se realizan actualizaciones periódicas (por ejemplo, nuevos medicamentos o cambios en protocolos sanitarios).
    - El mantenimiento garantiza que el sistema siga siendo seguro, legal y actualizado a lo largo del tiempo.

Ventajas

- Todo está **muy documentado y controlado**.
- Cada cambio está **registrado y aprobado**.
- Se garantiza la **seguridad y fiabilidad** del sistema.

Desventajas

- El proceso es **lento y costoso**.
- No se pueden hacer cambios fácilmente una vez comenzado.
- El usuario final (médico o enfermero) **no prueba nada hasta el final**, lo que puede provocar falta de flexibilidad o adaptación.

Conclusión

El modelo en cascada es el **más adecuado para proyectos grandes, críticos y estables**, donde:

- Los requisitos **no cambian**.
- Se requiere **seguridad, precisión y trazabilidad legal**.
- Un error podría tener **consecuencias graves** (como pérdida de información médica o errores clínicos).

Por eso se usa en:

- Sistemas médicos y hospitalarios.
- Software aeroespacial y de defensa.
- Sistemas bancarios y de control industrial.

El **modelo en cascada** es como construir un hospital real: primero se diseña todo sobre plano, se aprueba, y solo entonces se empieza a construir.
Si intentas cambiar la estructura cuando ya has levantado las paredes, el riesgo y el coste se disparan.

Ejemplo 2: Metodología Ágil – Aplicación de Reservas para un Restaurante

Proyecto:

Desarrollo de una **app móvil** que permita a los clientes:

- Reservar mesas en un restaurante desde el teléfono.
- Recibir confirmaciones automáticas.
- Dejar valoraciones y comentarios sobre la experiencia.

Se trata de un proyecto **dinámico y cambiante**, con **usuarios reales** que irán dando su opinión para mejorar la aplicación.

Características del proyecto

- Requisitos **no totalmente definidos** (el cliente quiere probar ideas nuevas).
    - El cliente tiene una idea general (quiere una app de reservas), pero aún no sabe exactamente qué funciones son más útiles.
    - Por eso el equipo de desarrollo debe ir **probando y adaptando** en función de la experiencia de los usuarios.
    - Los proyectos innovadores cambian sobre la marcha. No puedes esperar tenerlo todo planificado antes de empezar.
- Necesidad de **salir rápido al mercado** para competir con otras apps.
    - Hay **competencia**: otras apps similares ya existen.

  - Si el proyecto se desarrolla durante muchos meses sin mostrar resultados, puede quedar **obsoleto** antes de lanzarse.
  - Por eso se busca lanzar **una versión mínima (MVP)** lo antes posible para **empezar a competir** y **mejorar con el tiempo**.
  - En mercados rápidos, la clave no es la perfección inicial, sino **aparecer pronto y evolucionar continuamente**.

- Retroalimentación continua del **cliente y de los usuarios**.
  - Cada pocas semanas, el equipo presenta avances, el cliente los prueba y propone ajustes.
  - Así, el producto **evoluciona en base a la experiencia real**, no solo a una planificación teórica.
  - Este ciclo evita grandes errores, porque los problemas se detectan pronto y se corrigen antes de avanzar.

Aplicación de la metodología ágil (Scrum)

Scrum organiza el trabajo en ciclos cortos llamados sprints, normalmente de 2 a 3 semanas, donde el equipo se centra en lograr una pequeña parte funcional del producto.

Cada sprint incluye planificación, desarrollo, revisión y mejora.
Además, hay reuniones diarias breves (daily scrum) donde el equipo comenta:

- Qué hizo ayer.
- Qué hará hoy.
- Qué dificultades ha encontrado.

1. **Sprint 1: Diseño básico + función de reservar mesa.**
    - Se crea la **estructura inicial** de la app (pantalla principal y botón de reserva).
    - Se lanza una primera versión sencilla (**MVP**, producto mínimo viable) solo con esa función.
    - El objetivo es tener algo real que el cliente pueda probar, aunque sea muy básico.
2. **Sprint 2: integración con notificaciones y base de datos.**
    - La app ahora permite guardar las reservas en una base de datos y enviar **notificaciones automáticas** (confirmación, recordatorio, cancelación).
    - El cliente prueba y comenta qué mensajes o colores prefiere.
    - Se añade valor funcional sin esperar a tener todo terminado.
3. **Sprint 3: sistema de valoraciones y mejora del diseño.**
    - Se incluye una función para que los usuarios **dejen su valoración** y se optimiza la interfaz.

El cliente observa cómo reaccionan los usuarios y sugiere mejoras visuales.

- Las mejoras surgen de la experiencia de uso, no de la teoría.

4. **Sprint 4:** optimización de rendimiento y lanzamiento final.
    - Se corrigen errores, se mejora la velocidad y se publica la versión 1.0 en tiendas de aplicaciones.
    - Después, se sigue mejorando con actualizaciones (1.1, 1.2, etc.).
    - En ágil, el proyecto no termina con el lanzamiento: **siempre está en evolución.**

Cada sprint dura 2-3 semanas, con **reuniones diarias breves (daily scrum)** y **feedback del cliente** tras cada entrega.

Resultado

Producto disponible en el mercado **en pocas semanas**.
Los usuarios prueban la app y dan feedback real.
Los cambios se incorporan fácilmente entre ciclos.
Requiere **disciplina, comunicación constante** y **cliente involucrado**.

Ventajas:

- El producto está **en el mercado en pocas semanas**.
- Los usuarios **prueban la app real** y dan **feedback continuo**.
- Los cambios se incorporan **sin grandes costes**.
- El cliente **participa activamente** en el desarrollo.

Desventajas:

- Requiere **mucha comunicación y organización** entre el equipo y el cliente.
- Si el cliente no participa, el proceso pierde sentido.
- Sin disciplina, puede convertirse en **caos o improvisación**.

Conclusión

La metodología **ágil es más eficiente** cuando se busca **velocidad, adaptabilidad y mejora continua**.
Ideal para proyectos **dinámicos y competitivos** como apps móviles, startups o productos en evolución constante.

Mientras que el modelo en cascada diseña un edificio completo antes de construirlo, la metodología ágil empieza levantando una planta y, con la opinión de quienes la usan, va construyendo el resto.

Criterio	Cascada (Sistema hospitalario)	Ágil (App de reservas)
Tipo de proyecto	Crítico, normativo y estable	Innovador, cambiante y competitivo
Tiempo de entrega	Largo (producto al final)	Corto (entregas parciales frecuentes)
Flexibilidad ante cambios	Muy baja	Muy alta
Participación del cliente	Ocasional	Constante
Riesgo de obsolescencia	Alto (producto puede quedar desfasado)	Bajo (producto evoluciona continuamente)
Eficiencia global	Alta en entornos regulados	Alta en entornos dinámicos