Metodologías ágiles

Entornos de desarrollo

Diapositivas realizadas por Aitor Ventura Edo

Manifiesto Ágil

🎯 Los 4 valores (con hábitos concretos)

> **Resumen en una frase:** Agilidad = **personas**, **software funcionando**, **colaboración** y **adaptación al cambio** para entregar **valor temprano y continuo**.

👥 **Personas e interacciones** → *mejor feedback y decisiones más rápidas*

- **Haz:** dailies para **planificar el día**, *pair programming*, canales abiertos.
- **Evita:** handoffs innecesarios, “manda un correo y ya”.

✅ **Software funcionando** → *progreso visible*

- **Haz:** demos **quincenales**, *feature flags*, MVP (Producto Mínimo Viable).
- **Evita:** documentación eterna sin nada que probar.

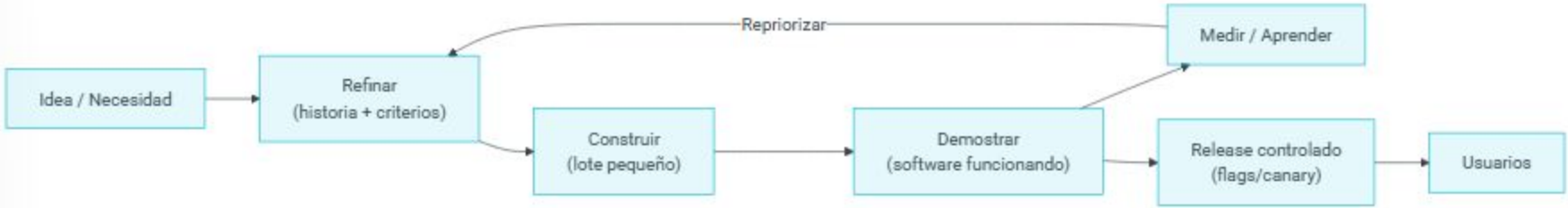
🤝 **Colaboración con el cliente** → *alineación de valor*

- **Haz: refinamiento** con *stakeholders*, criterios de aceptación conjuntos.
- **Evita:** contratos rígidos que ignoran datos de uso.

⮂ **Respuesta al cambio** → *decidir con información reciente*

- **Haz:** re-priorizar por **impacto/medición** cada semana.
- **Evita:** “seguimos el plan aunque ya no tenga sentido”.

De idea a valor: ciclo ágil con feedback



Cada vuelta reduce incertidumbre y evita invertir en la dirección equivocada.

🧩 Los 12 principios, agrupados para recordar mejor

🔥 Entrega temprana y frecuente (1, 3, 7)

1) Satisfacer al cliente con **entregas tempranas y frecuentes**.
3) Entregar **software funcionando** a menudo (semanas, no meses).
7) Medir el progreso por **software funcionando**.

Colaboración y aprendizaje continuo (2, 4, 6, 11, 12)

2) Aceptar **cambios** incluso tarde.

4) Negocio y desarrollo **juntos** a diario.

6) Conversación **cara a cara** (o remoto equivalente).

11) Equipos **autoorganizados**.

12) **Inspección y adaptación** periódicas (retros).

Calidad y sostenibilidad (5, 8, 9, 10)

5) Personas **motivadas** y confiables.

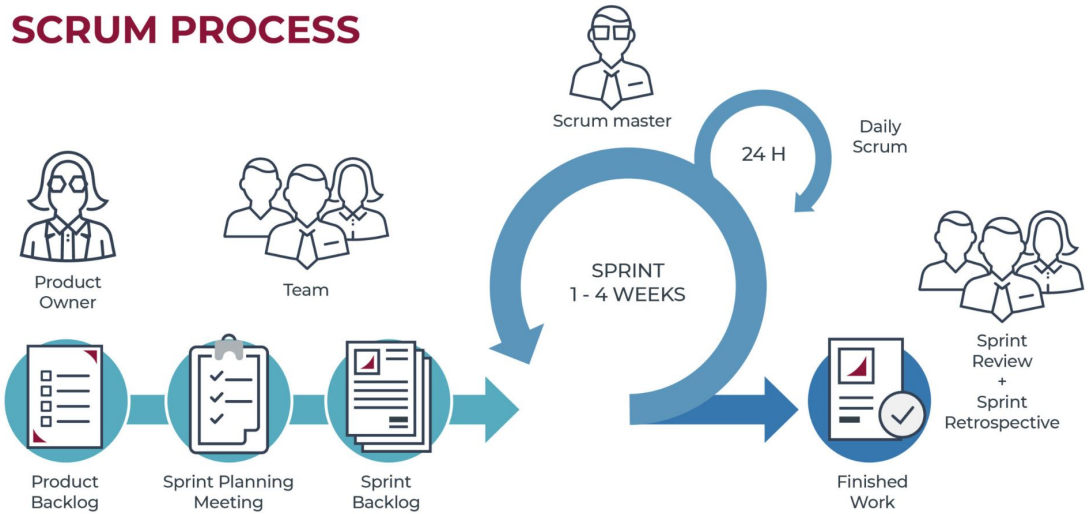
8) Ritmo **sostenible**.

9) **Excelencia técnica** y buen diseño.

10) **Simplicidad**: maximizar el **trabajo no hecho**.

Scrum: roles, artefactos, eventos; DoR/DoD

Se basa en principios iterativos e incrementales para gestionar proyectos de manera flexible y colaborativa.



Roles de Scrum

En Scrum intervienen diferentes actores, a cada actor se le puede asociar un rol concreto.

- **Product owner:** responsable de definir y priorizar los requisitos del producto. El Product Owner toma decisiones sobre qué funcionalidades incluir en el producto y en qué orden se desarrollarán.

Roles de Scrum

- **Scrum master:** facilita el proceso Scrum y ayuda al equipo a eliminar obstáculos para maximizar su productividad. El Scrum Master también se asegura de que el equipo siga las prácticas de Scrum y mejore continuamente su rendimiento.

- **Equipo de desarrollo:** profesionales encargados de convertir los requisitos en funcionalidades del producto final. El equipo de desarrollo es autónomo y autoorganizado, y se compromete a completar las tareas seleccionadas durante el sprint.

Reuniones de Scrum

Durante el proceso de Scrum se realizan diferentes reuniones en las que intervienen algunos de los roles mencionados.

- **Sprint planning:** Al comienzo de cada sprint, el equipo se reúne en una sesión de Sprint Planning para seleccionar las tareas que se abordarán durante el sprint. El Product Owner presenta los elementos del backlog que considera prioritarios, y el equipo de desarrollo estima el esfuerzo necesario para completar cada tarea. Al final de esta reunión, el equipo tiene un plan claro para el sprint.

Reuniones de Scrum

- **Daily Scrum:** Durante el sprint, el equipo se encuentra diariamente en la reunión Daily Scrum, también conocida como la reunión **de** seguimiento diario. En esta reunión corta, cada miembro del equipo comparte brevemente qué hizo ayer, qué planea hacer hoy y si ha encontrado algún obstáculo. El objetivo es mantener a todos informados y ayudar a eliminar obstáculos para la productividad.

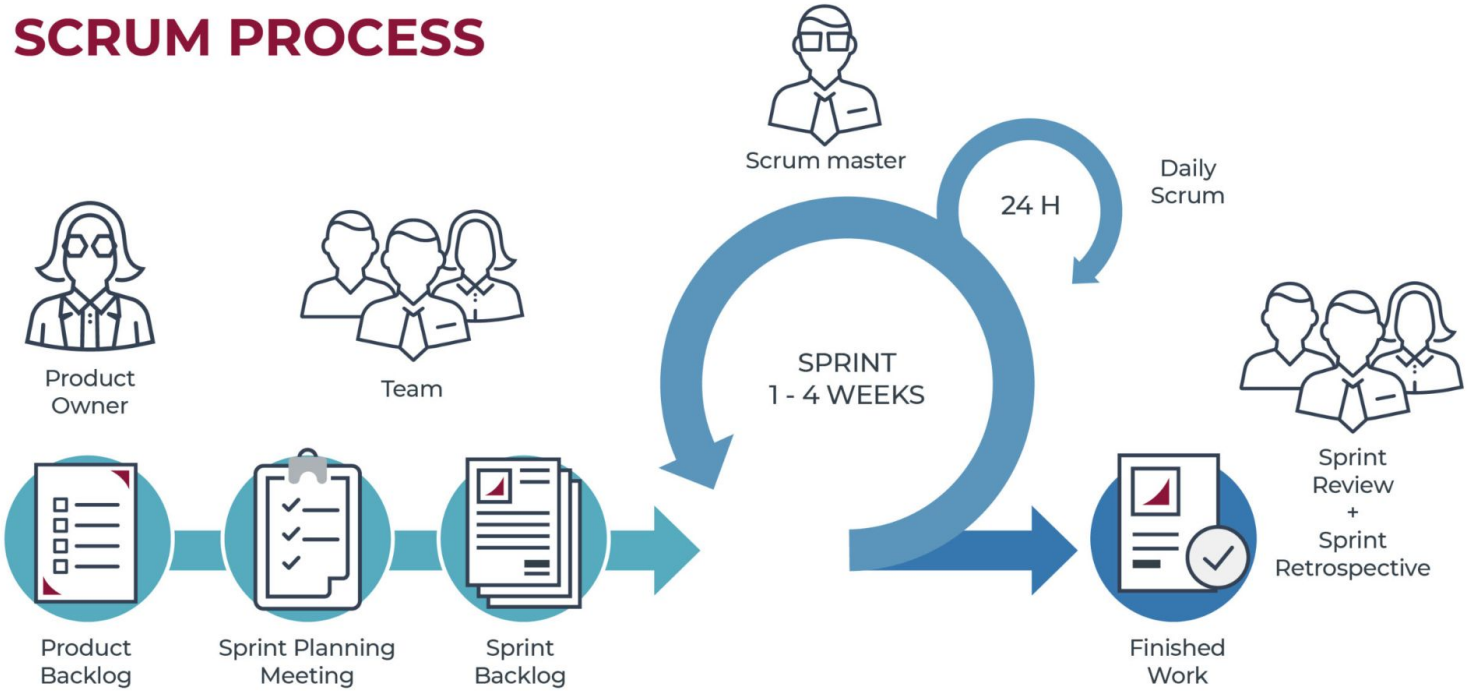
Reuniones de Scrum

- **Sprint Review:** Al finalizar el sprint, el equipo realiza una Sprint Review para demostrar las funcionalidades completadas al Product Owner y a otros stakeholders. Durante esta reunión, se obtiene retroalimentación valiosa sobre el trabajo realizado y se pueden realizar ajustes en el producto en función de los comentarios recibidos.

Reuniones de Scrum

- **Sprint Retrospective:** Después de la Sprint Review, el equipo lleva a cabo una Sprint Retrospective para reflexionar sobre el sprint. En esta reunión, se discuten qué salió bien, qué se podría mejorar y se identifican acciones específicas para el próximo sprint. La Sprint Retrospective fomenta la mejora continua y el trabajo en equipo.

SCRUM PROCESS



👥 Roles

- **Product Owner (PO):** define **Product Goal**, prioriza el **Product Backlog** y maximiza valor.
- **Scrum Master (SM): facilita** Scrum, elimina **impedimentos** y promueve mejora.
- **Equipo de desarrollo: multidisciplinar y autoorganizado**, construye el **Incremento**.

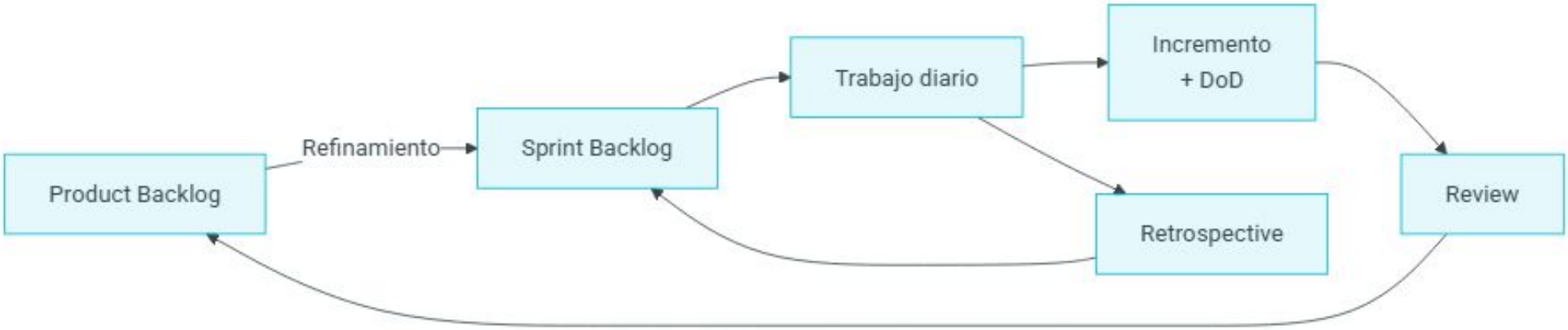
📦 Artefactos

- **Product Backlog (PB):** lista **ordenada por valor** (historias, épicas, bugs).
- **Sprint Backlog (SB):** selección para el sprint + **Sprint Goal** + plan.
- **Incremento:** suma de PB **terminado** al final del sprint y potencialmente **liberable**.

> **Transparencia:** cada artefacto incluye **medidas de calidad** (DoD) y, opcionalmente, métricas de previsibilidad (p. ej., *forecast*).

🗓️ Eventos (time-boxes típicos con sprint de 2 semanas)

- **Sprint (2 semanas):** contenedor de todo lo demás.
- **Sprint Planning (≈ 4 h):** qué (objetivo), por qué (valor) y cómo (plan inicial).
- **Daily Scrum (15 min):** sincronización y plan del día (**no** es reporte al jefe).
- **Sprint Review (≈ 2 h):** mostrar **Incremento** y **adaptar** el PB con *stakeholders*.
- **Retrospective (≈ 1.5 h):** mejora del **proceso**, **personas** y **herramientas**.
- **Refinement (actividad continua):** detallar PB (no es evento formal en la guía).


Product Backlog
Refinamiento
Sprint Backlog
Trabajo diario
Incremento
+ DoD
Review
Retrospective

✅ DoR / DoD (Definition of Ready / Definition of Done)

- **DoR (lista para entrar al sprint):**
  - Historia **clara** (Como/Quiero/Para), **criterios de aceptación**, **valor** entendido, **estimación** aproximada, dependencias visibles.
- **DoD (terminado de verdad):**
  - Código **mergeado**, **tests** pasados, **revisión** realizada, **documentación** mínima actualizada, desplegable (**incremento potencialmente liberable**).

> **Tip:** DoR/DoD son **políticas del equipo**; revísalas cuando la realidad cambie.

Kanban

Kanban gestiona el **flujo** de trabajo y mejora la **capacidad de entrega** con **políticas explícitas** y límites **WIP** (Work In Progress).

Reglas mínimas ¶

- **Visualiza** el flujo (tablero simple **To-Do / En curso / Revisar / Hecho**).
- Define **políticas** de movimiento entre columnas.
- **Limita WIP** (máximo de tareas simultáneas por columna/equipo).

XP: TDD, pair programming, refactorización continua

> **Idea clave:** XP = **excelencia técnica** para **cambiar con seguridad**: tests primero, código simple, revisiones constantes y entregas pequeñas.

🦴 Prácticas esenciales (de un vistazo)

🧪 TDD (Test-Driven Development)

- Escribe una **prueba que falla** → código **mínimo** → **refactor** sin romper.
- Beneficio: **diseño limpio** y **feedback** rápido.

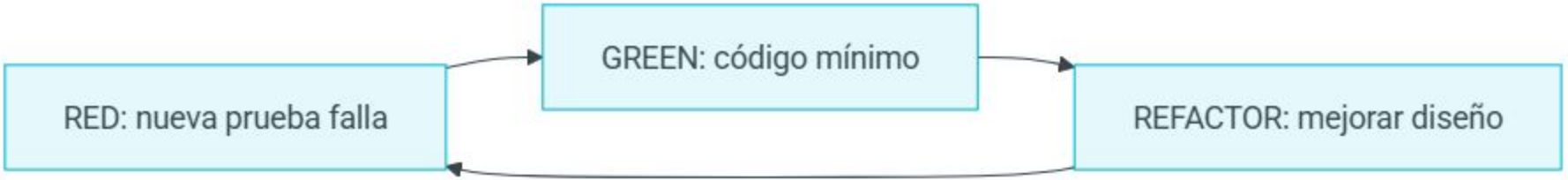
> **Idea clave:** XP = **excelencia técnica** para **cambiar con seguridad**: tests primero, código simple, revisiones constantes y entregas pequeñas.

🦴 Prácticas esenciales (de un vistazo)

🧪 TDD (Test-Driven Development)

- Escribe una **prueba que falla** → código **mínimo** → **refactor** sin romper.
- Beneficio: **diseño limpio** y **feedback** rápido.

🔁 TDD en 10 segundos



Mini-ejemplo (Java)

Primero la prueba (roja) → luego el código (verde) → por último, refactor.

Java

```java
// RED: test falla
@Test void totalConIVA() {
  assertEquals(121.0, Precio.conIVA(100.0, 0.21));
}

// GREEN: implementación mínima
class Precio {
  static double conIVA(double base, double iva) { return base * (1 + iva); }
}

// REFACTOR: nombres y redondeo
static double conIVA(double base, double iva) {
  return Math.round(base * (1 + iva) * 100.0) / 100.0;
}
```

👥 Pair programming

- Dos personas, un teclado: **Driver** (teclea) / **Navigator** (piensa estrategia).
- Alterna roles y usa **Ping-Pong TDD** (test ↔ código).

🔧 Refactorización continua

- Mejora **nombres**, **duplicidades** y **acoplamientos sin** cambiar comportamiento.
- Apoya en tests para moverte **con seguridad**.

Modos de *pair programming*

- **Driver/Navigator:** alterna cada 15–20 min o por test.
- **Ping-Pong TDD:** A escribe test → B lo hace pasar → B escribe test → A lo hace pasar…
- **Strong-Style:** quien **tiene la idea, no teclea** (obliga a explicar el diseño).
- **Mob (3+):** útil en piezas complejas; temporizador y turnos cortos.

Consejos prácticos

- Acuerda un **estilo de commits** (atómicos, mensaje claro).
- Usa un **timer** para los relevos (p. ej., 15 min).
- Checklist breve de **revisión** al cerrar la sesión de *pair*.

Integración continua (CI)

- **Commits pequeños** → build + **tests** + **análisis estático** siempre verdes.
- Evita ramas largas; integra **a diario**.

Propiedad colectiva

- El código es del **equipo** (no de una persona). Estándares, *lint*, **DoD**.

Lean Software Development y waste

> **Idea clave:** Lean = **eliminar desperdicio**, **aprender rápido** y **optimizar el sistema de extremo a extremo** para entregar **valor continuo**.

🧭 Principios Lean (aterrizados a software)

Eliminar desperdicio

- Quita lo que **no aporta valor** al usuario (colas, burocracia, re-trabajo).

Calidad desde el origen

- Tests automatizados, *linters*, *pairing* y *CI*; "**para la línea**" si falla (*andon*).

🔁 Aprender y mejorar

- Entregas pequeñas + métricas → *feedback* rápido → ajuste del plan.

⏳ Decidir tarde, entregar pronto

- Mantén opciones abiertas y **compromete** cuando hay datos; *feature flags* y *canary*.

💞 Respeto por las personas

- Equipos **multifuncionales** y autonomía para mejorar el flujo.

🔀 Optimizar el todo

- Mira el **flujo end-to-end** (no sólo “mi módulo”).

🗑️ Los 7 desperdicios (y cómo cazarlos)

Desperdicio (Lean)	En software	Señal de alarma	Contra-medida rápida
Trabajo parcialmente hecho	Ramas eternas, *features* a medias	Mucho "En curso"	**WIP** bajo, *feature flags*, entregas pequeñas
Funcionalidad extra	Oro molido que nadie usa	Uso bajo de nuevas *features*	Descubrir primero (MVP), medir adopción
Reaprendizajes/transferencias	Handoffs innecesarios	Muchas idas y vueltas	Equipos **multifuncionales**, DoR claro

Cambios de tarea	Multitarea	Context switching alto	Lotes pequeños, foco por **sprint/kanban**
Esperas	Aprobaciones, entornos, terceros	Tareas “bloqueadas”	Políticas explícitas, *self-service* de entornos
Defectos	Bugs y retrabajo	Alto % de retrabajo	TDD, *shift-left* (QA temprano)
Procesos extra	Burocracia que no añade valor	Mucho papeleo	Simplificar *workflow*, automatizar
Talento infrautilizado *(añadido común en software)*	Personas sin voz	Ideas se pierden	Retro efectiva, Kaizen con acciones

Herramientas ágiles: tableros y roadmaps

> **Objetivo:** que el **tablero** haga fluir el trabajo (reglas claras, límites de trabajo y métricas) y que el **roadmap** sea una **brújula de resultados** (qué impacto buscamos), no una lista de tareas.

🗂️ Tableros (cuándo usar cada uno)

Jira

- **Útil para:** equipos **grandes** o con procesos formales.
- **Ventajas:** permisos detallados, estados a medida, automatizaciones.

Trello

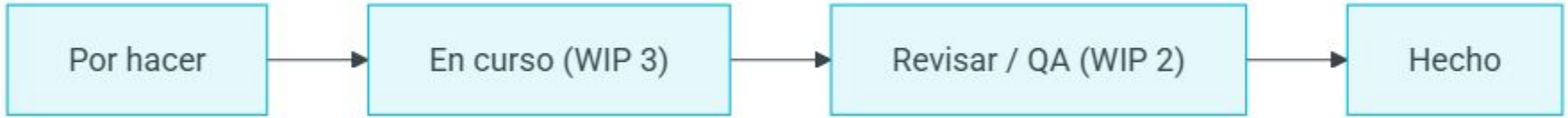
- **Útil para: equipos pequeños**, formación y proyectos ligeros.
- **Ventajas:** muy **visual** y fácil de empezar; listas y checklists.

GitHub Projects

- **Útil si ya usas GitHub para el código.** Conecta tareas con **issues** (incidencias) y **pull requests** (solicitudes de cambio).
- **Ventajas:** trazabilidad técnica natural; vistas por **tabla**, **tablero** y **cronograma**.

🧱 Plantilla de columnas (sirve en cualquier herramienta)

- `Por hacer → En curso (WIP n) → Revisar/QA (WIP m) → Hecho`



Pista: deja visibles los **límites WIP** (cuántas tarjetas caben) y revisa cuellos de botella cada semana.

Políticas claras (ejemplo mínimo)

- A **En curso** ⇒ hay **capacidad libre** y la tarjeta cumple **DoR** (está entendida y acotada).
- A **Revisar/QA** ⇒ hay **pull request**, **pruebas en verde** y *lint* pasado.
- A **Hecho** ⇒ cumple **DoD** (fusionado, probado, documentación mínima, *flag* si aplica).

📊 Métricas que debería “emitir” tu tablero

- **Lead time** (desde que se **pide** algo hasta **Hecho**).
- **Cycle time** (desde que **empieza** hasta **Hecho**).
- **Throughput** (cuántas tarjetas se terminan por semana).
- **% de tarjetas bloqueadas** y **tiempo bloqueado**.
- **CFD** (*Cumulative Flow Diagram* = diagrama de flujo acumulado) para ver estabilidad.

> 🔥 **Rutina semanal (15’): revisión de salud del flujo**
>
> - Mira el **CFD**, columnas con **WIP alto**, motivos de **bloqueo** y acordad **1 acción** de mejora.