

UNIVERSIDAD NACIONAL DE INGENIERÍA

Ingeniería de Software

Procesos, planificación y estimación

Cascada, Scrum y Kanban · Story points y velocity · El Boeing 737 MAX

Gerson Garrido

Semana 2

Hoy

Sesión A

1. Qué es un proceso de desarrollo
2. Cascada, iterativo e incremental
3. El Manifiesto Ágil
4. Scrum según la Scrum Guide 2020
5. Kanban y la ley de Little

Sesión B

6. Estimar no es prometer
7. Story points y planning poker
8. Velocity y planificación de un sprint
9. Caso: el Boeing 737 MAX
10. Laboratorio: el backlog de Kallpa

Lectura

The Scrum Guide (2020) y el Manifiesto Ágil con sus doce principios · informe del Congreso de EE. UU. sobre el 737 MAX, resumen ejecutivo (págs. 4–33) · Mountain Goat Software, *Planning Poker* y *What Are Story Points?*

SECCIÓN 01

Qué es un proceso

01

Un proceso es un acuerdo

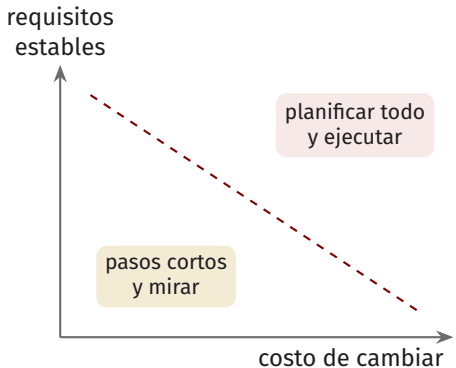
Cómo pasa el trabajo de una idea a software en uso. Todo proceso responde:

1. ¿Quién decide qué se construye?
2. ¿En qué orden?
3. ¿Cada cuánto se entrega?
4. ¿Cómo se corrige el rumbo?

"Nosotros no tenemos proceso"

También responden las cuatro, sin haberlo conversado: decide quien está cerca del teclado, el orden es el de los mensajes de WhatsApp, se corrige cuando el cliente se queja.

Cada proceso es una apuesta



- ¿Qué tan estables son los requisitos?
- ¿Qué tan caro es cambiar?
- Detrás está el **riesgo**; la herramienta es la **retroalimentación**.

La pregunta útil

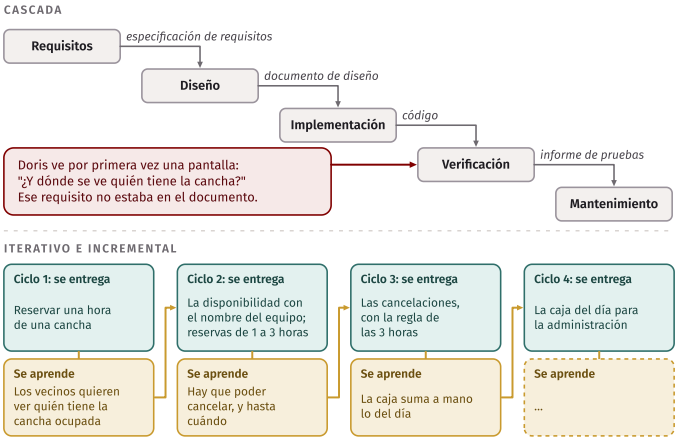
¿Cada cuánto y de quién recibimos retroalimentación, y a qué costo?

SECCIÓN 02

Cascada, iterativo e incremental

02

La cascada



Si la respuesta de Doris es no, se perdió un ciclo, no seis meses.

La cascada en Kallpa

- Cada documento se aprueba antes de pasar a la fase siguiente; volver atrás es una excepción.
- Kallpa así: dos meses de requisitos, seis meses en total. Al sexto mes Doris ve la primera pantalla: *"¿Y dónde se ve quién tiene la cancha?"*

Cuándo funciona la cascada

Funciona si

- Los requisitos casi no cambian (una norma).
- Corregir tarde es aceptable.

Su riesgo

El cliente ve el producto **al final**, cuando cambiar es lo más caro.

Error común: “Royce la recomendó”

Royce (1970) dibuja el modelo secuencial y lo llama riesgoso: las pruebas llegan al final. Propone diseñar antes, documentar, *hacerlo dos veces*, planificar las pruebas e involucrar al cliente.

Iterativo e incremental

Incremental

Se entrega por partes; cada entrega agrega algo que **se puede usar**.

Iterativo

Se avanza en ciclos; cada ciclo **vuelve sobre lo hecho** con lo aprendido.

	Se entrega	Se aprende
1	Reservar una hora	Quieren ver quién tiene la cancha
2	Nombre del equipo; 1 a 3 horas	Hay que poder cancelar
3	Cancelaciones (3 horas antes)	La caja suma a mano
4	La caja del día	…

Cada entrega es una pregunta: *"¿esto es lo que necesitaban?"*

Dividir la cascada en meses ("mes 2: diseño") no es iterativo: el cliente no puede probar nada.

SECCIÓN 03

El Manifiesto Ágil

03

Cuatro valores (2001)

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

La frase que más se olvida

"Aunque valoramos los elementos de la derecha, valoramos más los de la izquierda." El README de Kallpa es documentación, y es útil.

Algunos de los doce principios

- Entrega temprana y continua de software con valor.
- Aceptar que los requisitos cambien, incluso tarde.
- Entregar software funcional “entre dos semanas y dos meses”.
- “El software funcionando es la medida principal de progreso.”
- Ritmo sostenible, “de forma indefinida”.
- Atención continua a la excelencia técnica.
- Simplicidad: maximizar el trabajo *no* realizado.
- Reflexionar a intervalos regulares y ajustar.

Error común

“Ágil” como excusa para no planificar, probar ni documentar.

Y no es un método

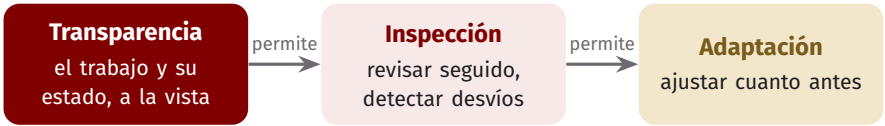
No dice cuánto dura un ciclo ni quién ordena el trabajo.

SECCIÓN 04

Scrum según la Scrum Guide 2020

04

Empirismo: tres pilares



- El conocimiento viene de la experiencia y de decidir con lo observado; *lean thinking*: menos desperdicio.
- "Inspeccionar sin adaptar se considera inútil."
- Guía oficial de Schwaber y Sutherland: versión de **noviembre de 2020**, menos de 13 páginas.

El Scrum Team: tres responsabilidades

Un equipo, “típicamente de 10 personas o menos”, sin subequipos ni jerarquías; multifuncional y autogestionado.

Product owner

Maximiza el valor; **ordena** el product backlog. Una persona, no un comité.

Scrum master

Que Scrum se use bien; quitar impedimentos. No es el jefe.

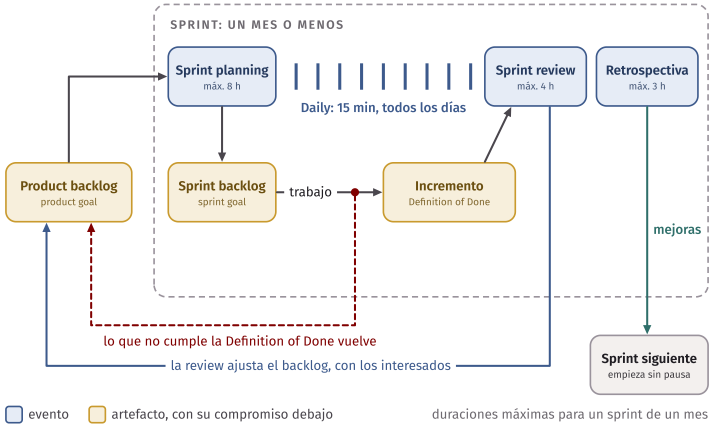
Developers

Crean el incremento: el plan del sprint y la calidad (Definition of Done).

Cambio de 2020

Accountabilities, no “roles”; ya no hay un “equipo de desarrollo” aparte: un solo equipo, un product goal.

Los cinco eventos



Los cinco eventos: para qué sirve cada uno

Evento	Para qué	Máximo (sprint de un mes)
Sprint planning	Por qué (sprint goal), qué y cómo	8 h
Daily	Avance hacia el sprint goal; plan del día	15 min, todos los días
Sprint review	Resultado con los interesados; qué sigue	4 h
Retrospectiva	Cambios para mejorar calidad y efectividad	3 h

Tres detalles que se olvidan

- **La daily es de los developers.** Eligen el formato; se enfoca en el sprint goal y termina con un plan. Las tres preguntas (“¿qué hice ayer?…”) se quitaron en 2020.
- **La sprint review no es una presentación**: es una sesión de trabajo con los interesados que ajusta el backlog.
- **La retrospectiva termina con cambios**, incluso en el sprint backlog del sprint siguiente.
- Solo el product owner puede cancelar un sprint, y solo si su sprint goal quedó obsoleto.

Error común: ceremonias vacías

Una daily que es un informe al jefe, una retrospectiva sin acciones: todo el costo de Scrum, nada de su beneficio.

Tres artefactos, tres compromisos

Artefacto	Qué es	Compromiso
Product backlog	Lista ordenada y emergente de lo que hace falta; la única fuente de trabajo	Product goal
Sprint backlog	Sprint goal (por qué) + ítems (qué) + plan (cómo); de los developers	Sprint goal
Incremento	Paso concreto hacia el product goal; verificado y usable	Definition of Done

Un buen sprint goal

Un resultado (“que los vecinos puedan cancelar desde la página”), no una lista de tickets.

La Definition of Done

Definition of Done

La descripción formal del estado que debe tener el incremento para cumplir las medidas de calidad del producto.

Un ítem que no la cumple

No se entrega ni se muestra en la review. **Vuelve al product backlog.** No existe “al 90 %”.

Para Kallpa, por ejemplo:

- `make check` pasa (tipos, linter, pruebas).
- Hay pruebas de cada criterio de aceptación.
- El pull request tiene una revisión aprobada.
- El README está al día si cambió una regla o la API.
- Está en `main`.

Cada punto se puede comprobar. “Código de calidad” no se puede.

Lo que la Scrum Guide no dice

Busquen en la guía:

- story points
- velocity
- user story
- planning poker
- Definition of Ready

No aparecen. Son prácticas complementarias.

Burn-downs

Aparecen una vez, como práctica de pronóstico: útiles, pero “no reemplazan la importancia del empirismo”.

Scrum es “inmutable”

Se pueden usar partes, “pero el resultado no es Scrum”.

Scrum en el proyecto del curso

P1	**Sprint 1 · P2**	**Sprint 2 · P3**	**Sprint 3 · P4**
requisitos	demo en la sem. 8	demo en la sem. 11	demo en la sem. 14

- Sprints de unas tres semanas; la demo al cliente hace de sprint review.
- `docs/sprints/sprint-N.md`: planificación (objetivo, historias, quién, capacidad), revisión y retrospectiva con acciones y responsables.
- Conviene que alguien del equipo asuma la responsabilidad del product owner: ordenar el backlog con lo que dijo el cliente.

SECCIÓN 05

Kanban

05

Kanban: la guía de 2025

Una estrategia para optimizar el flujo de valor. Tres prácticas:

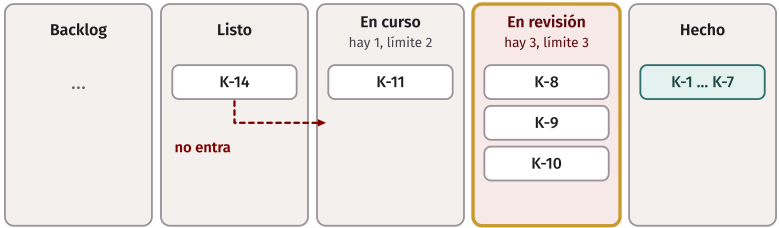
1. **Definir y visualizar** el flujo (*Definition of Workflow*).
2. **Gestionar activamente** los ítems: controlar el WIP, que no envejezcan, desbloquear.
3. **Mejorar** el flujo, cuando haga falta.

Sin sprints, sin roles fijos, sin eventos obligatorios. De Toyota; para software, desde Corbis (2006).

La Definition of Workflow dice

qué es un ítem; cuándo empieza y cuándo termina; sus estados; cómo se controla el WIP; las políticas de cada paso; y una **SLE**: “el 85 % se termina en 8 días o menos”.

Limitar el trabajo en curso



En revisión está en su límite: nadie empieza nada nuevo, aunque En curso tenga lugar.

Quien termine lo suyo ayuda a revisar.

- **Sistema pull**: lo nuevo entra cuando hay una señal de capacidad. Terminar vale más que empezar.
- La guía exige controlar el WIP; el límite por columna es una forma, no la definición.
- En GitHub Projects el límite **no impide** agregar tarjetas: solo lo resalta. La regla la cumple el equipo.

Cuatro métricas de flujo

WIP ítems empezados y no terminados

Throughput ítems terminados por unidad de tiempo (se cuentan ítems, no puntos)

Work item age tiempo desde que empezó un ítem que todavía no termina

Cycle time tiempo entre que empezó y terminó

Contamos días completos: empezar y terminar el mismo día es cycle time 1.

```
node flow-demo.ts

día  1  2  3  4  5  6  7  8  9 10
K-1   #  #  #  #  .  .  .  .  .  .   cycle time 4
K-2   #  .  .  .  .  .  .  .  .  .   cycle time 1
K-3   .  #  #  #  #  .  .  .  .  .   cycle time 4
K-4   .  .  #  #  #  #  #  #  #  .   cycle time 7
K-5   .  .  .  #  #  #  #  #  .  .   cycle time 5
K-6   .  .  .  .  #  #  #  .  .  .   cycle time 3
K-7   .  .  .  .  .  #  #  #  #  #   cycle time 5
K-8   .  .  .  .  .  .  .  #  #  #   cycle time 3
K-9   .  .  .  .  .  .  .  .  #  #   edad 2
WIP   2  2  3  4  4  4  4  4  4  3
```

La edad es la única que avisa antes: con una SLE de 6 días, una edad de 5 merece atención hoy.

La ley de Little

Ley de Little (teoría de colas, 1961)

En un sistema estable, en promedio:

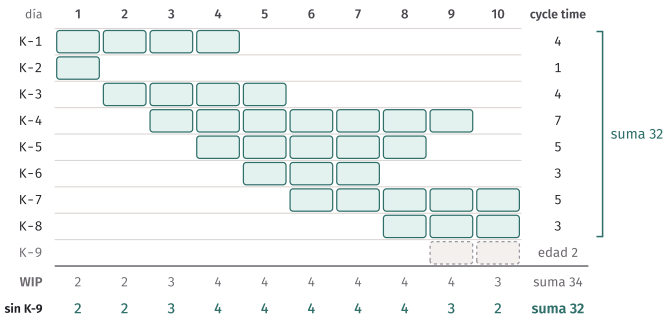
$$\textbf{WIP} = \textbf{throughput} \times \textbf{cycle time} \qquad \text{(o sea, cycle time} = \text{WIP / throughput)}$$

```
node flow-demo.ts (continúa)

Solo los ítems terminados (K-1 a K-8):
  WIP promedio 3.20 | throughput 0.80 por día | cycle time promedio 4.00 días
  throughput × cycle time = 3.20
Con K-9, que sigue en curso:
  WIP promedio 3.40 | throughput 0.80 por día
```

- Exacta si la ventana empieza y termina vacía: cada día en curso suma 1 al WIP y 1 al cycle time.
- Con trabajo que entra y no sale (K-9), ya no es exacta. Habla de promedios.
- No está en la Kanban Guide, pero explica por qué limitar el WIP importa.

Contar los mismos cuadrados de dos formas



Cada cuadrado suma 1 a su fila y 1 a su columna: contados de las dos formas, K-1 a K-8 dan 32.

WIP promedio	throughput	cycle time promedio	ley de Little
32 / 10 días = 3.2	8 ítems / 10 días = 0.8	32 / 8 ítems = 4	0.8 × 4 = 3.2

Mismo throughput, más WIP: más cycle time

```
node flow-demo.ts (continúa)

Simulación: el equipo termina 2 ítems por día; días 11 a 30
  límite  4: WIP promedio 4.00 | throughput 2.00 | cycle time promedio 2.00 días
  límite  6: WIP promedio 6.00 | throughput 2.00 | cycle time promedio 3.00 días
  límite 10: WIP promedio 10.00 | throughput 2.00 | cycle time promedio 5.00 días
```

- El equipo no trabaja más rápido ni más lento: solo tiene más cosas abiertas.
- La simulación fija la capacidad a propósito. En un equipo real, mucho WIP suele costar además cambios de contexto.

Error común

Confundir throughput (ítems, sin estimar) con velocity (puntos, en un sprint). O respetar los límites y no mirar la edad.

¿Cuál elegir?

	Cascada	Iterativo	Scrum	Kanban
Requisitos	Fijos	Se descubren	Se descubren; el PO ordena	Llegan sin parar
Se entrega	Al final	Cada ciclo	Cada sprint ($\leq$ 1 mes)	Cuando algo termina
Se corrige	Cambio formal	Cada ciclo	Review y retro	Revisando el flujo
Conviene	Estándar fijo	Hay un cliente que prueba	Cliente que revisa; un objetivo por ciclo	Trabajo sin aviso

- **El proyecto del curso**: un cliente, una demo por sprint $\Rightarrow$ Scrum.
- **El soporte de Kallpa en producción**: la página no carga, un mensaje que cambiar, una vulnerabilidad $\Rightarrow$ Kanban.
- Mezclar está bien si saben qué parte viene de dónde y por qué la usan.

Con IA

- Una herramienta acorta escribir el código.
- No acorta entender qué construir, revisar, probar ni desplegar. Muchas veces ese trabajo crece.
- Producir más rápido algo que nadie validó es llegar más rápido al error: la retroalimentación frecuente importa **más**.

En el tablero

Si la columna que se llena es *En revisión*, el límite de WIP les recuerda que el cuello de botella ya no es escribir.

SECCIÓN 06

Estimar no es prometer

06

Estimación, compromiso y objetivo

Estimación

Lo que creemos que cuesta, con su incertidumbre.

"Entre dos y tres sprints."

Compromiso

Lo que el equipo acepta entregar.

"Para la demo, ver el precio antes de reservar."

Objetivo

Lo que el negocio quisiera.

"El campeonato, para el 5 de enero."

Lo que se negocia

El **alcance** o el **riesgo**, no la estimación. Bajar el número sin cambiar el trabajo es fabricar una fecha imposible.

McConnell, *Software Estimation: Demystifying the Black Art* (2006).

SECCIÓN 07

Story points y planning poker

07



Story points

Story point

Una estimación **relativa** del **esfuerzo** para terminar un ítem, hasta cumplir la Definition of Done.

Entran tres cosas a la vez:

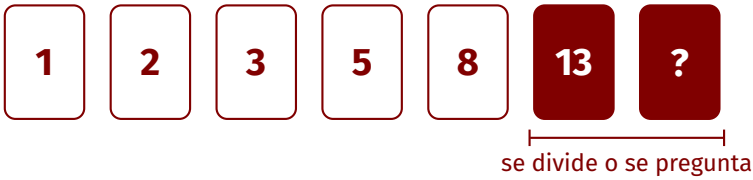
- cantidad de trabajo,
- complejidad,
- riesgo e incertidumbre.

Relativos: si "filtrar las canchas por deporte" es un 2, "cobrar los feriados como fin de semana" podría ser un 3.

¿Por qué no horas?

- Las horas dependen de *quién*.
- Comparar es más fácil que medir en absoluto.
- La conversión a calendario se mide (velocity), no se adivina.

Una escala con huecos



- Nadie distingue un 6 de un 7: la escala no deja fingir esa precisión.
- **13 o “?”**: demasiado grande o incierto para un sprint.
- Fowler: si la carta más alta quiere decir “13 o más”, usarla en un pronóstico engaña.

Error común

“1 punto = 1 día”: son horas con otro nombre. Y sumar puntos de equipos distintos: cada equipo tiene su escala.

Planning poker

1. El product owner presenta el ítem.
2. El equipo pregunta y aclara qué es “terminado”.
3. Cada persona elige en secreto.
4. Todos muestran a la vez.
5. Si están lejos: explican **la más alta y la más baja**, y se vuelve a votar.

Estiman los developers. El product owner explica y responde, pero no estima por el equipo.

Por qué funciona

Evita el **anclaje** al primer número que se dice, y da voz a quien habla menos.

En el laboratorio

“Lejos” es **más de un salto**: 3 y 5, no; 3 y 8, sí.

Un ejemplo: “V2 Inscribir a un niño en una clase”

	Ana	Beto	Carla	Diego
Primera votación	2	3	3	8
Segunda votación	5	5	5	8

- 2 y 8: tres saltos. **Ana**: “Es un formulario; la clase ya existe”.
- **Diego**: “No hay tabla de inscripciones: migración en `database.ts`, ruta nueva, errores en el README, pruebas HTTP. ¿Y si inscriben dos veces al mismo niño?”
- 5 y 8: un salto. Se anota **5**, la más votada, y la pregunta para el cliente.

No se promedia

Promediar 2 y 8 da 5, un número que nadie votó y que esconde lo que valía la pena entender.

La regla como código

poker.ts (extracto)

```ts
export const CARDS = [1, 2, 3, 5, 8, 13] as const;
export type Card = (typeof CARDS)[number];
export type Vote = Card | '?';

export function needsDiscussion(votes: readonly
    Vote[]): boolean {
  if (votes.includes('?')) return true;
  const cards = numeric(votes);
  return stepsBetween(Math.min(...cards) as Card,
                      Math.max(...cards) as Card) >
      1;
}
```

Lo que comprueban las pruebas

`needsDiscussion([2,3,3,8])`	`true`
`needsDiscussion([5,5,5,8])`	`false`
`agreedEstimate([5,5,5,8])`	`5`
`agreedEstimate([2,3,3,8])`	`undefined`

Una unión de literales (`as const`) en lugar de un `enum`: Node solo quita los tipos.

Dividir un pedido grande

"Vacaciones útiles: clases de vóley y básquet para niños en enero y febrero." Primera votación: **13, 13, ?, ?**.

V1	Publicar horarios y cupos	2
V2	Inscribir a un niño	5
V3	Cerrar al llenarse el cupo	2
V4	Lista para el profesor	1
V5	Cobrar la mensualidad	5
V6	Descuento por hermanos	3
V7	Avisar si se suspende	3
	Total	**21**

- Slices verticales que se entregan y prueban solos, no "la base de datos" y "la página".
- **¿Suman 13?** No: al dividir aparece trabajo que no se veía. Lo normal es que sumen más.
- El 13 original quería decir "demasiado grande", no 13.
- Dividir abre la priorización: ¿V6 y V7 pueden esperar? (Semana 3.)

SECCIÓN 08

Velocity y planificación

08

Velocity

```
node plan-demo.ts

Sprint 3 al cierre:
  hecho     1  Mostrar el reglamento en la página
  hecho     3  Preguntas frecuentes en la página
  hecho     5  Calendario de la semana de una cancha
  no hecho  3  Avisar por WhatsApp si el complejo cierra un día
  velocity: 9 puntos
```

Velocity

La suma de los puntos de los ítems **terminados** en un sprint, según la Definition of Done.

- Medición después: no se “fija”.
- 0 % o 100 %: lo a medio hacer no suma.
- Del equipo; no se compara entre equipos.

Planificaron 12. Al aviso le faltaba la revisión: no suma.

Pronosticar con un rango

```
node plan-demo.ts (continúa)

Velocity de los sprints cerrados: 8, 11, 9 (promedio 9.3)
Vacaciones útiles: 21 puntos -> entre 2 y 3 sprints
```

- 21 / 11 $\approx$ 1.9 $\Rightarrow$ 2 sprints.
- 21 / 8 $\approx$ 2.6 $\Rightarrow$ 3 sprints.
- 21 / 9.3 $\approx$ 2.3 parece más preciso. No lo es.

El pronóstico supone

el mismo equipo, la misma escala, la misma Definition of Done y un alcance que no crece.

Agile Alliance: el pronóstico con velocity suele ser acertado, pero rara vez preciso.

La sprint planning

¿Por qué? el sprint goal · **¿Qué?** ítems según desempeño y capacidad · **¿Cómo?** tareas de un día o menos

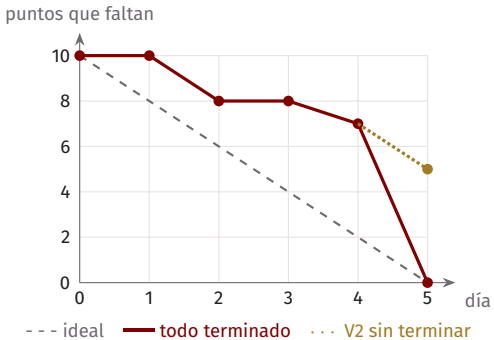
```
node plan-demo.ts (continúa)

Primera propuesta para un sprint de 10 puntos:
  entra    2  V1 Publicar horarios y cupos de las clases
  entra    5  V2 Inscribir a un niño en una clase
  entra    2  V3 Cerrar la inscripción al llenarse el cupo
  queda    5  V5 Cobrar la mensualidad y verla en la caja
  entra    1  V4 Lista de inscritos para el profesor
  queda    3  V6 Descuento por hermanos
  queda    3  V7 Avisar al apoderado si se suspende una clase
  total: 10 puntos
```

`fillSprint` sigue el orden del product owner y salta lo que no cabe (V5). Es una propuesta: manda el sprint goal.

Sprint goal: “que los apoderados puedan ver las clases e inscribir a sus hijos hasta llenar el cupo”.

El burndown chart



- Un ítem baja la curva solo el día en que cumple la Definition of Done: V1 el día 2, V4 el 4, V2 y V3 el 5.
- Las mesetas son normales; una curva plana hasta el último día, no.
- Si V2 no termina: quedan 5, la velocity es 5 y V2 vuelve al backlog.
- Sirve para conversar a tiempo con el product owner. No reemplaza el empirismo.

La velocity no es una meta

Premiar al equipo con más puntos, o exigir que la velocity suba:

- Los puntos los estima el propio equipo $\Rightarrow$ el 3 pasa a ser 5.
- La velocity sube sin entregar nada más.
- Y se pierde la medida que servía para planificar.

Ley de Goodhart

Goodhart (1975), sobre indicadores monetarios. La forma con que se cita es de Strathern (1997): *"cuando una medida se convierte en meta, deja de ser una buena medida"*.

Primero la pregunta: Goals/Signals/Metrics

Goal: el resultado, sin decir cómo se mide · **Signal**: cómo sabríamos que lo logramos · **Metric**: lo que sí se mide, como aproximación. Cada métrica se rastrea hasta su objetivo.

Goal	Signal	Metric
Lo que pide la administración llega rápido	Los pedidos se terminan poco después de empezarse	Cycle time de *En curso* a *Hecho*
Lo que se entrega funciona	Pocos errores llegan a los vecinos	Issues `bug` por semana; pruebas que fallan en `main`
Ritmo sostenible	Nadie trabaja de madrugada para cerrar	Encuesta breve en cada retrospectiva

Solo vale la pena medir si una decisión concreta depende del resultado (SWEG, cap. 7).

Con IA: estimen el trabajo completo

- La herramienta escribe la función en un minuto.
- Entenderla, escribir las pruebas de los criterios de aceptación y atender la revisión: una hora.
- El ítem cuesta esa hora. Se estima hasta la Definition of Done.

DORA 2025

La IA como **amplificador** de las fortalezas y debilidades que la organización ya tiene.

Basado en encuestas: describe asociaciones, no prueba causas.

SECCIÓN 09

Caso: el Boeing 737 MAX

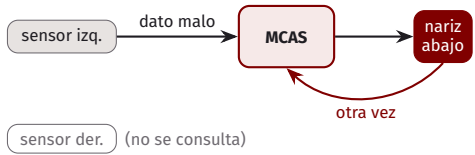
09

Qué pasó

- **29 oct. 2018**: Lion Air 610, Yakarta; 189 muertos.
- **10 mar. 2019**: Ethiopian 302, Adís Abeba; 157 muertos.
- Dos Boeing 737 MAX, certificados en marzo de 2017. 346 muertos.

MCAS

Software que baja automáticamente la nariz del avión en ciertas condiciones. Decidía con un solo sensor de ángulo de ataque, aunque el avión tiene dos.



Los pilotos, en su mayoría, no sabían que el sistema existía.

El informe: cinco temas centrales

1. **Presión de producción**: competir con el A320neo; recortar costos, mantener el calendario.
2. **Supuestos de diseño equivocados**: un solo sensor; pilotos que manejarían la falla de un sistema que no conocían; el MCAS no se clasificó como crítico para la seguridad.
3. **Cultura de ocultamiento**: hacia la FAA, los clientes y los pilotos.
4. **Conflicto de interés**: empleados de Boeing que actuaban en nombre de la FAA.
5. **Influencia de Boeing** sobre la supervisión de la FAA.

Qué es esta fuente

Una investigación legislativa de 18 meses, preparada por el personal de la mayoría del comité, con documentos internos, entrevistas y audiencias. No es la investigación técnica de los accidentes (Indonesia, Etiopía).

Un objetivo de negocio convertido en requisito

- Contrato con Southwest (dic. 2011): descontar al menos 1 millón de dólares por avión si la FAA exigía entrenamiento en simulador.
- 200 pedidos firmes.
- Evitar el simulador se volvió una **directiva del programa**, e influyó en cómo se describió el MCAS ante la FAA.

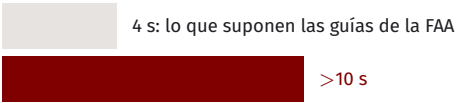
Lo de esta sesión

Objetivo $\neq$ compromiso $\neq$ estimación. Aquí, un objetivo comercial se impuso sobre lo que decían los datos.

Un cambio de alcance y un supuesto sobre las personas

Marzo de 2016

- Se rediseña el MCAS: actúa también a baja velocidad, con más autoridad.
- Horas después, la FAA aprueba quitar el MCAS del manual de los pilotos. Quienes lo aprueban **no saben del rediseño**.
- El ingeniero jefe que aprobó el rediseño no sabía que usaba un solo sensor ni que se activaba repetidamente.



Boeing no compartió el dato con la FAA ni con sus clientes.

Una alerta que no funcionaba

- La alerta *AOA Disagree* avisa cuando los dos sensores no coinciden. Debía ser estándar.
- El software la ató a un indicador **opcional**: no funcionaba en más del 80 % de los aviones.
- Boeing lo probó en julio de 2015 y no lo detectó; lo supo en agosto de 2017 y siguió entregando aviones sin informar.

Error común: "fue un bug de software"

El MCAS hizo lo que su diseño decía, y ese diseño contradecía los requisitos internos de Boeing (no interferir con el pilotaje). Sí hubo un defecto, el de la alerta. Pero la causa de los accidentes no fue un error de programación: fallaron los requisitos, el análisis de riesgo, la revisión independiente y la honestidad sobre lo que se sabía.

Lo que el caso le dice a este curso

Semanas 3 y 4	Un supuesto sobre los pilotos sin verificar; un requisito de alto nivel que el diseño contradecía
Semanas 9 y 10	Un defecto en software probado, entregado durante casi dos años
Semana 11	Un rediseño que no pasó por quien debía revisarlo; revisores con conflicto de interés
Semana 15	Saber algo grave y no decirlo
Esta semana	Un objetivo comercial por encima de lo que decían los datos

En un proyecto común

Una fecha que obliga a dejar de probar; un cambio “pequeño” sin mirar qué más toca; un supuesto sobre los usuarios que nadie comprobó; un problema conocido que no se reporta.

SECCIÓN 10

Laboratorio

10

El backlog de Kallpa en GitHub

```bash
$ gh auth login
$ gh label create precios --color FBCA04 --description "Tarifas y descuentos"
$ gh issue create --title "Ver el precio antes de reservar" \
    --body-file pedido.md --label funcionalidad --label precios
```

`code/week-02/backlog.md`: 15 pedidos, claros y vagos, pequeños y enormes.

1. Etiquetas y un issue por pedido; lo vago, como preguntas.
2. Tablero con límites 2 y 3 y un campo *Puntos*.
3. Definition of Done en un issue fijado.
4. Planning poker; dividir lo que salga 13 o “?”.
5. Sprint de 10 puntos con su sprint goal. ¿Y con Kanban?

Con IA

Si una herramienta redacta los issues, revísenlos contra el pedido: suele rellenar huecos con requisitos que nadie dijo.

make test

code/week-02

```
$ make test
tsc -p .
eslint .
node --test "*.test.ts"
✓ hay exactamente 15 pedidos, numerados del 1 al 15 (0.7075ms)
...
✓ con un ítem todavía en curso al final, la igualdad ya no es exacta (0.095458ms)
✓ con el mismo throughput, más WIP significa más cycle time (2.188458ms)
...
✓ el ejemplo del apunte: 2, 3, 3, 8 y, después de conversar, 5, 5, 5, 8 (0.118ms)
...
ℹ tests 26
ℹ suites 0
ℹ pass 26
ℹ fail 0
```

Las afirmaciones de la clase, como pruebas; `make demo` imprime las salidas de las diapositivas.

Para la próxima clase

- Leer la Scrum Guide completa y el Manifiesto con sus doce principios (Sesión A).
- Leer el resumen ejecutivo del informe sobre el 737 MAX y las dos lecturas de Mountain Goat Software (Sesión B).
- Traer su repositorio de Kallpa de la semana 1 y `gh` funcionando.
- P0 se entrega la próxima semana.

Pregunta

En la última reunión de su grupo de un curso anterior, ¿quién decidía qué se hacía, en qué orden, y cuándo se corregía el rumbo? ¿Qué proceso era, aunque nadie le pusiera nombre?