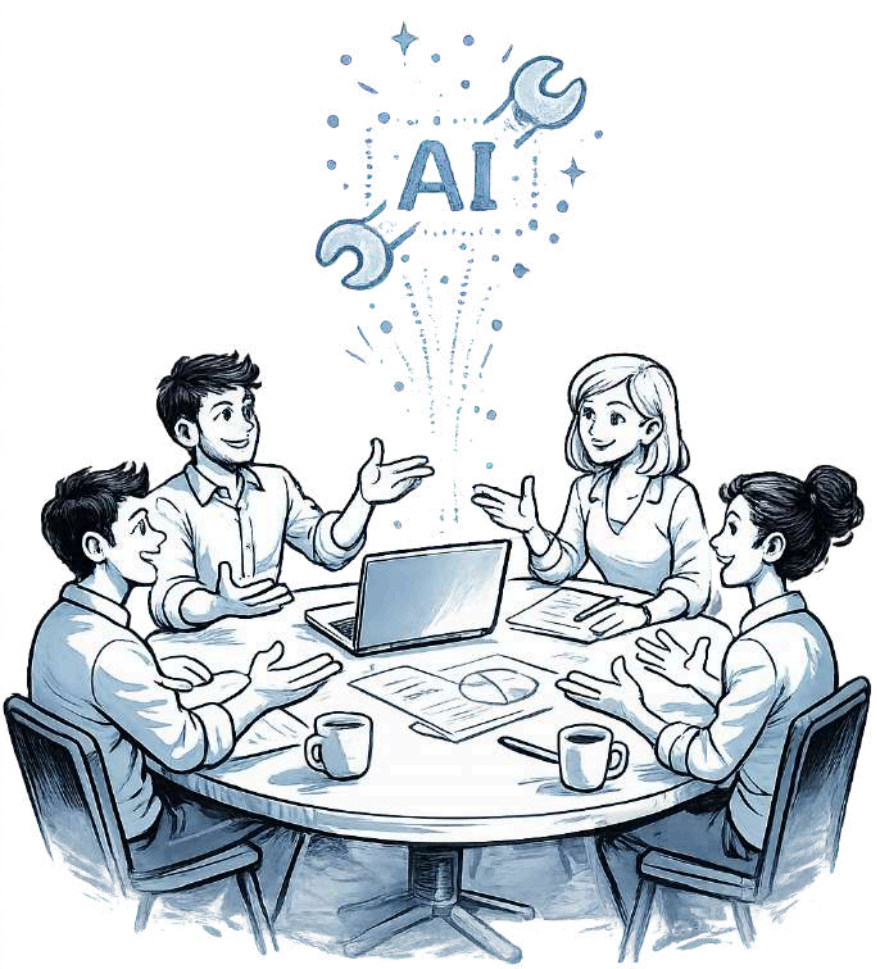
Scrum en equipos con IA

Adapta tu práctica ágil cuando la inteligencia artificial transforma la forma de crear productos



Guía Didáctica

Versión 1.0 — Marzo 2026

Scrum en equipos con IA

Adapta tu práctica ágil cuando la inteligencia artificial transforma la forma de crear productos

Versión: 1.0 – marzo 2026

© Autor: Juan Palacio Asistente de IA: Claude Opus 4.6 (Anthropic)

Edita: Scrum Manager (scrummanager.com)

Los contenidos de esta guía didáctica están sujetos a revisión y actualización.

Consulta siempre la versión más reciente en scrummanager.com.

Tabla de contenidos

Introducción

Existe una máxima ampliamente aceptada en la industria del software: cuando cambia la tecnología con la que se crea el producto, deben cambiar los modelos de gestión. La inteligencia artificial generativa ha alterado las condiciones de trabajo de los equipos de desarrollo. No se trata de un cambio incremental: es una transformación de las premisas básicas sobre las que se diseñaron los marcos de trabajo ágiles.

Los asistentes de código, los agentes autónomos y los modelos de lenguaje de gran escala han comprimido ciclos que antes requerían semanas en horas. Han permitido que perfiles no técnicos generen prototipos funcionales. Han desplazado el cuello de botella de la creación desde la ingeniería hacia la gestión de producto.

Y sin embargo, el 80% de las organizaciones que utilizan IA reportan no observar un impacto significativo en sus resultados de negocio. Los equipos producen más, pero no necesariamente entregan más valor. Esta paradoja —más velocidad, pero no más agilidad— es el punto de partida de esta guía.

Scrum, el marco de trabajo predominante en el desarrollo ágil, emergió en los años 80 como respuesta espontánea de equipos de ingeniería al modelo de cascada. Desde entonces no ha dejado de enriquecerse con las aportaciones de la comunidad profesional. Sus principios—empirismo, autoorganización, entrega iterativa de valor—siguen siendo profundamente relevantes. Sin embargo, las prácticas concretas necesitan ser revisadas a la luz de una realidad donde los equipos ya no son exclusivamente humanos y donde la velocidad de entrega se ha multiplicado exponencialmente.

Esta guía es un documento formativo, no normativo. No pretende definir un «nuevo scrum» ni establecer reglas que sustituyan a las que cada equipo haya adoptado. Su propósito es ofrecer un mapa actualizado del terreno para que cada profesional ágil tome sus propias decisiones informadas.

La idea central es esta: los principios ágiles son una brújula que no pierde su norte con la IA, pero las prácticas concretas necesitan adaptarse. Para cada área de tensión, presentamos opciones, evidencias y riesgos, respetando la autonomía de cada equipo para encontrar su propio camino.

Objetivos de aprendizaje

- Comprender cómo la IA generativa ha transformado las condiciones del desarrollo de productos.
- Distinguir qué principios ágiles permanecen invariables y qué prácticas necesitan evolucionar.
- Adaptar los roles de scrum master, product owner y desarrolladores al nuevo contexto.

- Evolucionar artefactos y eventos para equipos que trabajan con IA.
- Conocer los modelos emergentes: doble carril, Spec-Driven Development, IA como teammate.
- Implementar mecanismos de calidad y gobernanza para código generado por IA.
- Facilitar la conversación de adaptación en tu equipo u organización.

Cómo usar esta guía

Esta guía está estructurada en ocho capítulos que pueden leerse de forma secuencial o consultarse de manera independiente según las necesidades.

Los dos primeros capítulos establecen el contexto y los fundamentos: qué ha cambiado con la IA y por qué los principios ágiles siguen siendo la brújula. Los capítulos 3 a 5 abordan la evolución de los elementos clásicos de scrum: roles, artefactos y eventos. Los capítulos 6 y 7 presentan los modelos emergentes y los mecanismos de calidad. El capítulo final ofrece herramientas para liderar la evolución en tu equipo.

Cada capítulo incluye:

- Conceptos clave con evidencia de la industria.
- Conexiones con los documentos de la comunidad ágil (*Scrum Guide Expansion Pack*, *AI Teammate Framework*, etc.).
- Ejercicios de reflexión para aplicar a tu contexto.
- Casos prácticos ambientados en equipos reales.
- Preguntas de autoevaluación.

Te recomendamos tener a mano información sobre tu equipo actual: su tamaño, sus prácticas, su nivel de adopción de IA, sus principales fricciones. Los ejercicios serán más valiosos si los aplicas a tu situación real.

> **Nota importante**
>
> La integración de la IA en equipos ágiles es un campo en evolución rápida. Los modelos y prácticas que aquí se presentan reflejan el estado del conocimiento a principios de 2026. La mejor forma de usar esta guía es como punto de partida para la experimentación, no como receta definitiva

Formación y acreditación

Los contenidos de esta guía están disponibles en Skill Arena, la plataforma de autoformación y entrenamiento de Scrum Manager. Allí puedes realizar ejercicios de práctica, refuerzo y evaluación, y también obtener un diploma acreditativo del conocimiento adquirido. Como la aplicación de SDD con IA evoluciona rápidamente, Skill

Arena mantiene el contenido actualizado, por lo que es la referencia recomendada para consultar la versión más reciente.

CAPÍTULO 1 — El cambio de paradigma

La IA generativa no ha llegado como una mejora incremental. Ha alterado las reglas del juego de forma similar a como lo hizo internet en los años 90 o la computación en la nube en los 2000. Entender la naturaleza y profundidad de este cambio es el primer paso para adaptarse a él. En este capítulo exploraremos qué ha cambiado exactamente, por qué la adopción masiva de IA no se ha traducido automáticamente en resultados de negocio, y cuál es el debate actual en la comunidad ágil sobre cómo responder.

1.1. Qué ha cambiado: velocidad, democratización, equipos híbridos

La IA generativa ha introducido cambios profundos en la forma de crear productos digitales. No se trata de hacer lo mismo más rápido, sino de hacer cosas que antes no eran posibles.

Velocidad de creación

Los equipos pueden generar incrementos de producto, prototipos o funcionalidades en cuestión de horas o días. Lo que antes requería sprints completos puede ahora producirse en fracciones de ese tiempo. Un desarrollador con asistencia de IA puede producir en un día lo que antes le llevaba una semana. Un equipo pequeño puede abordar proyectos que antes requerían equipos mucho mayores.

Pero esta velocidad tiene matices importantes. La investigación de Harvard Business School encontró que desarrolladores senior experimentaron una disminución del 19% en su rendimiento en tareas complejas al usar IA, debido al tiempo dedicado a revisar y probar el output generado. La velocidad de generación no es lo mismo que la velocidad de entrega de valor.

Democratización de capacidad técnica

Un perfil de negocio puede hoy generar prototipos funcionales con herramientas de IA y nocode, sin intervención directa de ingeniería. Esto desdibuja la cadena de roles tradicional. El product owner puede visualizar ideas sin esperar a que el equipo las implemente. El diseñador puede generar código básico. El stakeholder puede crear mockups funcionales para comunicar lo que necesita.

Esta democratización tiene implicaciones profundas para la organización del trabajo. Las barreras entre roles se difuminan. Las dependencias tradicionales se reducen. Pero también emergen nuevos riesgos: más personas pueden crear más cosas, lo que amplifica la necesidad de validación y gobernanza.

Desplazamiento del cuello de botella

La capacidad de construir ya no es el factor limitante. La gestión de producto —decidir qué merece ser construido, validarlo con usuarios, mantener una visión estratégica— se ha convertido en el verdadero cuello de botella.

Esta es quizás la implicación más profunda para los equipos ágiles. Durante décadas, la ingeniería fue el recurso escaso. Las metodologías se optimizaron para maximizar el rendimiento del equipo de desarrollo. Ahora, la restricción está en otra parte: en la capacidad de decidir qué construir y validar que lo construido genera valor.

Equipos híbridos

El trabajo ya no es exclusivamente una interacción entre humanos. Los agentes de IA participan en la creación de código, la generación de tests, la revisión de arquitectura y la redacción de documentación. Las prácticas de gestión deben contemplar esta nueva realidad.

¿Es la IA un miembro del equipo? ¿Una herramienta sofisticada? ¿Un colaborador externo? Estas preguntas no tienen respuestas definitivas, pero sí tienen implicaciones prácticas para cómo organizamos el trabajo.

1.2. La paradoja de la productividad

Uno de los datos más reveladores del estado actual es lo que se ha denominado la «paradoja de la IA generativa»: aunque el 78% de las empresas utilizaban IA en 2025 y el 84% de los desarrolladores declaraban usar herramientas de codificación asistida, aproximadamente el 80% de las organizaciones reportaron no observar un impacto significativo en sus resultados de negocio.

¿Cómo es posible que tanta adopción genere tan poco impacto?

El diagnóstico

La brecha entre adopción y resultados se atribuye a un fenómeno concreto: los equipos producen funcionalidades a mayor velocidad, pero sin la guía de producto, la validación con usuarios ni la visión estratégica necesarias para que esa velocidad genere valor real.

Es como si un coche de carreras compitiera en un circuito sin mapa. Puede ir muy rápido, pero eso no garantiza que llegue a la meta.

Producir más no es ser más ágil

Esta paradoja es el argumento más sólido para revisar las prácticas ágiles. No basta con adoptar herramientas de IA; es necesario transformar cómo se gestiona el trabajo para que la velocidad técnica se traduzca en impacto.

La agilidad nunca fue velocidad. Es la capacidad de responder eficazmente al cambio mientras se entrega valor. La IA permite ir mucho más rápido, pero si esa velocidad no va acompañada de mecanismos de validación, reflexión y adaptación, el resultado es simplemente construir cosas equivocadas más rápidamente.

La pregunta que cambia

Cuando la IA permite crear features más rápido de lo que la organización puede validar su deseabilidad, la pregunta clave deja de ser «¿puede la IA construirlo?» y pasa a ser «¿deberíamos construirlo, y por qué?».

Este cambio de pregunta tiene implicaciones para todos los roles del equipo:

- **Para el product owner:** su trabajo se vuelve más estratégico y menos táctico.
- **Para el desarrollador:** su valor se desplaza de la implementación a la orquestación y la revisión.
- **Para el scrum master:** su foco se mueve de facilitar eventos a facilitar la reflexión sobre el proceso

Implicaciones prácticas

Los equipos que superan la paradoja de la productividad comparten características comunes:

- **Mantienen la validación al ritmo de la producción:** no permiten que la capacidad de construir supere la capacidad de validar. Si pueden generar diez features por sprint, se aseguran de poder validar diez features por sprint.
- **Invierten más en descubrimiento:** dado que la construcción es más barata, la proporción de esfuerzo se desplaza hacia entender qué construir.
- **Miden resultados, no outputs**: no celebran cuántas features entregaron, sino qué impacto tuvieron esas features en los usuarios y el negocio.

1.3. Scrum como conocimiento abierto: por qué puede evolucionar

Para entender cómo debe evolucionar scrum ante la IA, conviene primero entender qué es realmente scrum y cómo ha llegado hasta aquí.

Un origen comunitario, no propietario

Existe una confusión frecuente: muchos profesionales asumen que scrum es una metodología diseñada por autores específicos que definen qué es correcto y qué no. Esta visión es históricamente inexacta.

Scrum no es una metodología diseñada y difundida verticalmente desde una autoría académica, un comité de estandarización o un grupo de investigadores. A diferencia de modelos como CMMI (creado por la Universidad Carnegie Mellon) o SAFe (creado por Dean Leffingwell), scrum no tiene un propietario que lo defina con exclusividad.

En los años 80, los ingenieros de empresas como 3M, Fuji, Honda o HP, descontentos con los modelos de producción basados en ciclos de cascada, desarrollaban un patrón iterativo con solapamiento de fases. En 1986, Hirotaka Takeuchi e Ikujiro Nonaka identificaron estos principios en su artículo "The New New Product Development Game" y les dieron el nombre de «scrum». Es importante entender que Nonaka y Takeuchi identificaron, dieron nombre y visibilidad al concepto, pero no lo crearon.

Evolución continua

Desde entonces, la comunidad ágil ha continuado madurando y transformando las prácticas de scrum:

- La reducción de las iteraciones desde duraciones de uno o dos meses a una o dos semanas.
- La incorporación de las retrospectivas (que no estaban en el marco original).
- El refinamiento del backlog como práctica explícita.
- Las prácticas de estimación (y de no estimación).
- Los tableros kanban de gestión visual.

Todos son ejemplos de cómo la comunidad ha ido dando forma al scrum que conocemos hoy.

Por qué esto importa ahora

Esta perspectiva es fundamental. Si scrum fuera un modelo cerrado y propietario, adaptarlo a la IA requeriría esperar a que «sus autores» lo actualizaran. Pero si entendemos scrum como conocimiento abierto y emergente, la adaptación a la IA es exactamente lo que la comunidad profesional debe hacer: observar cómo ha cambiado el contexto y evolucionar las prácticas manteniendo los principios.

En junio de 2025, el *Scrum Guide Expansion Pack* de Sutherland, Coleman y Jocham declaró explícitamente que scrum es «mutable», un reconocimiento formal desde

dentro de la propia corriente de la *Scrum Guide* de que el marco debe adaptarse. Esta declaración no otorga permiso (que nunca fue necesario), pero sí valida lo que la comunidad ya venía practicando.

El espíritu que se mantiene

Lo que no cambia es el espíritu original: las prácticas deben ayudar a equipos a autogestionarse y mantener un flujo de avance continuo, produciendo resultados de forma iterativa y frecuente. Este espíritu es perfectamente compatible con la IA; de hecho, la IA puede potenciarlo si se usa bien.

1.4. El debate: ¿ruptura o evolución?

La comunidad ágil se encuentra dividida entre dos posturas ante el impacto de la IA. Ambas merecen ser comprendidas antes de decidir qué camino tomar.

La postura de la rotura

Sostiene que las prácticas estándar de scrum han llegado al fin de su efectividad. Los argumentos principales:

- Los ciclos de dos semanas resultan lentos ante la velocidad de la IA.
- Los equipos se reducen drásticamente, eliminando la necesidad de ceremonias de coordinación.
- Los roles clásicos están evolucionando o desapareciendo.
- El artefacto central (la historia de usuario) no es el formato adecuado para comunicarse con la IA.

Los defensores de esta postura abogan por nuevos paradigmas: vibe coding, vibe engineering, Spec-Driven Development, flujo continuo sin sprints. Algunos llegan a sugerir que «scrum ha muerto».

La postura de la evolución

Argumenta que la IA generativa es un multiplicador de fuerza para la agilidad, no su reemplazo. Los argumentos principales:

- Los principios ágiles (empirismo, adaptación, colaboración) son más críticos que nunca.
- La velocidad de la IA hace que la capacidad de reflexionar y validar sea más necesaria, no menos.
- Los eventos de scrum (especialmente la retrospectiva) son el mecanismo natural para inspeccionar cómo funciona la colaboración con la IA.
- El problema no es scrum; es cómo se ha implementado scrum.

La investigación de Harvard Business School encontró que individuos con IA igualaron el rendimiento de equipos enteros sin IA, y que equipos con IA fueron significativamente más propensos a producir soluciones de primera categoría. La evidencia respalda el impacto transformador de la IA, pero no necesariamente la muerte de los marcos ágiles.

Una tercera vía: evolución radical

Esta guía adopta una posición intermedia que podríamos llamar «evolución radical»:

- Los principios ágiles mantienen su vigencia.
- Las prácticas concretas necesitan una transformación profunda (no ajustes cosméticos).
- El cambio debe hacerlo cada equipo según su contexto, no esperar a que alguien lo prescriba.

Esta posición reconoce que algo fundamental ha cambiado (coincide con la postura de ruptura en la magnitud del cambio), pero confía en que los principios ágiles proporcionan la brújula para navegar ese cambio (coincide con la postura de evolución en el valor de los fundamentos).

El riesgo de cada postura

- **La postura de ruptura** corre el riesgo de tirar al bebé con el agua del baño: descartar mecanismos valiosos (como la retrospectiva) en nombre de la velocidad.
- **La postura de evolución** corre el riesgo de subestimar la magnitud del cambio: aplicar parches superficiales a prácticas que necesitan repensarse desde cero.
- **La evolución radical** intenta evitar ambos riesgos: transformación profunda, guiada por principios probados.

Ejercicios prácticos

Ejercicio 1: diagnóstico del cambio en tu equipo

Reflexiona sobre cómo la IA ha afectado a tu equipo o contexto profesional:

1. **Velocidad:** ¿En qué tareas ha aumentado significativamente la velocidad de producción? ¿En cuáles no ha cambiado?
2. **Roles:** ¿Han cambiado las responsabilidades de algún rol? ¿Hay tareas que antes hacía una persona y ahora hace (o ayuda) la IA?
3. **Cuello de botella:** ¿Dónde está ahora el cuello de botella del equipo? ¿Sigue siendo la capacidad de construcción o se ha desplazado?
4. **Validación:** ¿Podéis validar con usuarios al ritmo que producís? Si no, ¿qué se está acumulando sin validar?

5. **Calidad:** ¿Habéis notado cambios en la calidad del código o del producto desde que usáis IA?

Comparte estas reflexiones con tu equipo en la próxima retrospectiva.

Ejercicio 2: análisis de la paradoja de la productividad

Si tu equipo ya usa IA, evalúa si estáis experimentando la paradoja de la productividad:

Datos a recopilar:

- ¿Cuántas features/historias entregábais antes de usar IA? ¿Cuántas ahora?
- ¿Ha aumentado proporcionalmente la satisfacción de los usuarios? ¿Los resultados de negocio?
- ¿Qué proporción del tiempo se dedica a construir vs. a validar/descubrir?

Señales de que estáis en la paradoja:

- ☐ Entregamos más features pero el impacto en métricas clave no ha cambiado.
- ☐ El backlog crece más rápido de lo que validamos lo entregado.
- ☐ Hemos reducido tiempo de discovery para aumentar tiempo de delivery.
- ☐ No sabemos si las features que entregamos están siendo usadas.

Si identificas señales de la paradoja, ¿qué podría cambiar para resolverla?

Ejercicio 3: tu posición en el debate

Reflexiona sobre tu posición personal ante el debate ruptura vs. evolución:

1. ¿Qué elementos de tu forma actual de trabajar con scrum crees que deben mantenerse?
2. ¿Qué elementos crees que necesitan cambiar profundamente?
3. ¿Hay alguna práctica que crees que debería eliminarse por completo?
4. ¿Hay alguna práctica nueva que crees que debería incorporarse?
5. En una escala del 1 al 10:
    - 1 = "scrum está bien como está, la IA es solo una herramienta más"
    - 10 = "scrum ha quedado obsoleto, necesitamos un marco completamente nuevo".

    ¿Dónde te sitúas?

Argumenta tu posición y prepárate para debatirla con colegas que tengan posiciones diferentes.

Resumen del capítulo

En este capítulo hemos explorado la naturaleza del cambio que la IA introduce en el desarrollo de productos:

- La IA ha transformado la velocidad de creación, ha democratizado la capacidad técnica, ha desplazado el cuello de botella hacia la gestión de producto, y ha creado equipos híbridos humanos-IA.
- La paradoja de la productividad muestra que el 80% de las organizaciones con IA no ven impacto en resultados. Producir más no es automáticamente entregar más valor.
- Scrum es conocimiento abierto y emergente, no un modelo propietario. Esto significa que la comunidad puede (y debe) adaptarlo al nuevo contexto.
- El debate entre ruptura y evolución tiene argumentos válidos en ambos lados. Esta guía adopta una posición de «evolución radical»: transformación profunda guiada por principios probados.

En el próximo capítulo exploraremos esos principios: qué permanece invariable y por qué los valores ágiles no solo sobreviven a la IA sino que se vuelven más relevantes.

Preguntas de autoevaluación

1. ¿Cuáles son los cuatro cambios principales que la IA generativa ha introducido en el desarrollo de productos?
2. ¿Qué es la 'paradoja de la productividad' y cuál es su causa principal?
3. ¿Por qué se dice que scrum es 'conocimiento abierto' y qué implicaciones tiene esto para su evolución?
4. Describe las diferencias entre la postura de 'ruptura' y la postura de 'evolución' ante el impacto de la IA.
5. ¿En qué sentido el cuello de botella se ha desplazado de la ingeniería a la gestión de producto?

CAPÍTULO 2 — Los principios ágiles como brújula

Antes de analizar qué debe cambiar, conviene anclar qué no debe cambiar. Los valores y principios del desarrollo ágil no solo sobreviven a la era de la IA: se vuelven más relevantes.

En este capítulo exploraremos cómo cada valor del *Manifiesto Ágil* se ve amplificado (no debilitado) por la IA, por qué la IA es un amplificador de lo que el equipo ya trae, y cuál es el riesgo más importante a evitar: confundir velocidad con agilidad.

2.1. Los valores del *Manifiesto* que se amplifican

En febrero de 2001, diecisiete profesionales del software se reunieron en Salt Lake City y redactaron el *Manifiesto Ágil*. Veinticinco años después, sus cuatro valores centrales no solo siguen vigentes: la IA los hace más críticos.

«VALORAMOS MÁS A LOS INDIVIDUOS Y SU INTERACCIÓN QUE A LOS PROCESOS Y LAS HERRAMIENTAS»

Este valor adquiere una nueva dimensión cuando las herramientas incluyen agentes de IA que generan código, escriben tests y proponen soluciones. La investigación muestra que las personas que usan IA reportan emociones más positivas y menor carga cognitiva, lo que les permite dedicar más atención al trabajo de relación y comunicación.

La IA libera tiempo mental. La pregunta es: ¿a qué dedica el equipo ese tiempo? Si lo dedica a más interacción humana de calidad —conversaciones de diseño, validación con usuarios, alineación estratégica—, el valor se amplifica. Si lo dedica a generar más features sin conversación, se pierde.

El principio mantiene su esencia: la conversación humana sigue siendo irremplazable. Los equipos que usan IA para el análisis de clientes tienen conversaciones más ricas, no menos.

«VALORAMOS MÁS EL SOFTWARE QUE FUNCIONA QUE LA DOCUMENTACIÓN EXHAUSTIVA»

Con la IA, es posible generar prototipos funcionales en horas. Esto refuerza el principio de entregar software que funciona como mecanismo de validación. ¿Por qué documentar extensamente cómo será algo cuando puedes construir una versión funcional y probarlo?

Pero emerge una paradoja interesante: las especificaciones detalladas —una forma de documentación— se vuelven esenciales para guiar a la IA. El Spec-Driven Development propone que la especificación sea el artefacto principal del que se deriva el código.

La clave está en que estas especificaciones son documentación ejecutable: no burocracia sino el lenguaje común entre humanos e IA. No es documentación para guardar en un cajón; es documentación que se ejecuta y produce software.

«VALORAMOS MÁS LA COLABORACIÓN CON EL CLIENTE QUE LA NEGOCIACIÓN CONTRACTUAL»

Este valor se vuelve más necesario cuando la IA puede construir features más rápido de lo que se pueden validar. Si la construcción es barata y rápida, el riesgo de construir lo equivocado aumenta. La única forma de mitigarlo es más colaboración con el cliente, no menos.

La IA amplifica tanto los aciertos como los errores en la comprensión del problema. Un equipo alineado con el cliente construirá más cosas correctas a mayor velocidad. Un equipo desalineado construirá más cosas equivocadas a mayor velocidad.

«VALORAMOS MÁS LA RESPUESTA AL CAMBIO QUE EL SEGUIMIENTO DE UN PLAN»

La IA hace operativamente viable lo que antes era un ideal. Detectar y responder al cambio requería un procesamiento de información enorme; ahora, la IA puede monitorizar patrones de comportamiento, sentimiento y conversaciones de soporte en tiempo real. Lo que requería ciclos de investigación trimestrales puede ocurrir ahora semanal o diariamente.

Pero la capacidad de detectar cambios es inútil si el equipo no tiene el hábito de responder a ellos. La IA proporciona el sensor; el equipo debe proporcionar la respuesta. Y responder al cambio requiere momentos de reflexión —exactamente lo que las retrospectivas proporcionan.

2.2. La IA como amplificador, no como sustituto

Una idea transversal emerge de toda la investigación reciente: la IA amplifica lo que el equipo ya trae. No es neutral; es un multiplicador de tendencias existentes.

Si el equipo tiene buenas prácticas de producto...

La IA las potencia. Un equipo que ya validaba con usuarios puede ahora validar más rápido. Un equipo con buena visión estratégica puede explorar más opciones. Un equipo con disciplina de calidad puede detectar problemas antes.

La IA en manos de un buen equipo es como dar un motor más potente a un conductor experto: llega más lejos, más rápido, con más control.

Si el equipo tiene carencias...

La IA las expone y las agrava. Un equipo que ya tenía problemas de validación ahora produce más features sin validar. Un equipo con deuda técnica acumula más deuda más rápido. Un equipo sin claridad estratégica genera más ruido, más rápido.

La IA en manos de un equipo con problemas es como dar un motor más potente a un conductor que no domina el vehículo: puede estrellarse más rápido.

La separación del juicio

Como se ha señalado desde diversos foros de la comunidad ágil: la mayor amenaza no es que la IA reemplace a los profesionales ágiles, sino que revele que ciertas organizaciones nunca necesitaron verdaderos profesionales ágiles, sino simplemente a alguien que gestionara una herramienta.

La IA separa a quienes aportan juicio, experiencia y capacidad de manejar la complejidad humana de quienes solo realizaban tareas mecánicas que la tecnología podía haber automatizado hace una década.

Esto tiene implicaciones profundas:

- Para los profesionales: el valor diferencial está en lo que la IA no puede hacer (juicio, empatía, visión, liderazgo).
- Para las organizaciones: invertir en desarrollo de talento es más importante, no menos.
- Para los equipos: las competencias «blandas» se convierten en competencias «críticas».

Qué amplifica exactamente la IA

En el contexto de un equipo ágil, la IA amplifica:

- La velocidad de ejecución (de ideas a código, de specs a prototipos).
- La capacidad de exploración (probar más opciones, más rápido).
- El acceso a información (análisis de datos, síntesis de documentación).
- La productividad individual (un desarrollador puede hacer lo de varios).

Pero no amplifica automáticamente:

- La claridad sobre qué construir.
- La conexión con las necesidades del usuario.
- La cohesión del equipo.
- La capacidad de reflexionar y aprender.

Estas últimas siguen dependiendo de las personas y sus prácticas.

2.3. El riesgo de confundir velocidad con agilidad

Un error frecuente —y peligroso— es asumir que ir más rápido equivale a ser más ágil. Esta confusión existía antes de la IA, pero la IA la amplifica dramáticamente.

Velocidad vs. Agilidad

La agilidad no es velocidad. Es la capacidad de responder eficazmente al cambio mientras se entrega valor.

Un equipo puede ser muy rápido y muy poco ágil: entrega features a gran velocidad, pero sin saber si son las correctas, sin capacidad de cambiar de rumbo cuando la evidencia lo requiere, sin momentos de reflexión para mejorar.

Un equipo puede ser relativamente lento y muy ágil: entrega menos features, pero las que entrega están validadas, responde rápidamente a cambios del mercado, mejora continuamente su forma de trabajar.

Lo que la velocidad sin agilidad produce

Los datos de la industria son elocuentes:

- La duplicación de código en proyectos asistidos por IA aumentó del 8% al 12%.
- La actividad de refactoring cayó drásticamente: los equipos producen código nuevo en lugar de mejorar el existente.
- En 2024, por primera vez, el código copiado o generado por IA superó al código refactorizado.
- La confianza de los desarrolladores en la precisión del código generado por IA cayó del 69% al 54%.

Estos datos sugieren que, sin controles deliberados, la IA acelera la acumulación de deuda técnica. El equipo va más rápido, pero el código se vuelve frágil, duplicado y costoso de mantener.

La velocidad que importa

No toda velocidad es igual. Hay velocidades que importan y velocidades que son ilusorias:

- **Velocidad que importa:**
  - Velocidad de validación (cuánto tardamos en saber si una idea funciona).
  - Velocidad de aprendizaje (cuánto tardamos en incorporar feedback).
  - Velocidad de adaptación (cuánto tardamos en cambiar de rumbo).
- **Velocidad que puede ser ilusoria:**
  - Velocidad de generación de código (si ese código no se usa o está mal).

- Velocidad de cierre de historias (si esas historias no entregan valor).
- Velocidad de features entregadas (si esas features no se adoptan).

Las preguntas que protegen

Para evitar la confusión entre velocidad y agilidad, un equipo puede hacerse regularmente estas preguntas:

- ¿Estamos entregando más valor o solo más código?
- ¿Podemos cambiar de rumbo cuando la evidencia lo requiere, o estamos comprometidos con el plan?
- ¿Estamos aprendiendo a la velocidad que producimos?
- ¿Nuestra calidad se mantiene o se degrada con la velocidad?
- ¿Los usuarios están más satisfechos o solo reciben más features?

Si las respuestas no son claramente positivas, la velocidad puede estar siendo contraproducente.

2.4. Qué no debe cambiar: empirismo, reflexión, colaboración

Los principios que sustentan scrum —y la agilidad en general— son invariantes que la IA no debería erosionar, sino reforzar.

Empirismo

Scrum se fundamenta en tres pilares: transparencia, inspección y adaptación. La IA no debilita estos pilares; los potencia. Con dashboards más inteligentes, análisis predictivo sobre el progreso hacia los objetivos del sprint y ciclos de retroalimentación más rápidos, los equipos tienen hoy más capacidad de inspeccionar y adaptar que nunca.

Pero esta capacidad solo genera valor si se ejerce con disciplina. El riesgo es que la velocidad de la IA supere la capacidad del equipo para reflexionar sobre lo que está construyendo. La IA genera datos; el equipo debe convertirlos en aprendizaje.

El empirismo no es automático. Requiere momentos deliberados de inspección (¿qué ha pasado?) y adaptación (¿qué cambiamos?). La retrospectiva es el mecanismo por excelencia para esto. Eliminarla en nombre de la velocidad es renunciar al empirismo.

Reflexión

La mejora continua requiere reflexión. La reflexión requiere tiempo. La IA permite producir más en menos tiempo, lo que libera tiempo para reflexionar. Pero la tentación es usar ese tiempo para producir aún más, no para reflexionar más.

Los equipos que prosperarán con la IA son los que protegen deliberadamente el tiempo de reflexión:

- Retrospectivas que se mantienen aunque «no haya tiempo».
- Momentos de pausa entre sprints para procesar lo aprendido.
- Conversaciones de diseño que no se acortan para «ir más rápido».

Colaboración humana

La IA puede facilitar la colaboración (resumir conversaciones, proponer agendas, detectar conflictos), pero no puede sustituirla. Las relaciones de confianza se construyen entre personas. Las decisiones difíciles requieren juicio humano. Los conflictos se resuelven con empatía, no con algoritmos.

Un equipo que delega demasiada colaboración a la IA pierde la cohesión que necesita para funcionar. Los daily stand-ups, las plannings, las retrospectivas... no son solo intercambios de información; son rituales que construyen equipo.

Autoorganización

Scrum confía en que los equipos son capaces de decidir cómo hacer su trabajo. La IA no cambia esto; lo hace más importante. Con más herramientas y más opciones, las decisiones sobre cómo organizarse son más complejas. Un equipo que espera que alguien le diga exactamente cómo usar la IA está renunciando a la autoorganización.

- La autoorganización en la era de la IA incluye decidir:
- Qué tareas delegamos a la IA y cuáles hacemos los humanos.
- Qué nivel de revisión aplicamos al output de la IA.
- Cómo integramos la IA en nuestro flujo de trabajo.
- Qué límites ponemos al uso de la IA.

Estas son decisiones del equipo, no de la herramienta.

Ejercicios prácticos

Ejercicio 1: auditoría de valores ágiles

Evalúa cómo tu equipo vive actualmente los cuatro valores del Manifiesto Ágil, y cómo la IA está afectando a cada uno:

1. Individuos e interacciones vs. procesos y herramientas:
    - ¿El uso de IA ha aumentado o reducido la calidad de las conversaciones en el equipo?
    - ¿Dedicáis el tiempo liberado por la IA a más interacción humana?
2. Software funcionando vs. documentación extensiva:

    - ¿Usáis la IA para producir prototipos funcionales más rápido?
    - ¿Las especificaciones que usáis con la IA son documentación viva o burocracia?
3. Colaboración con el cliente vs. negociación contractual:
    - ¿La velocidad de la IA os ha acercado o alejado del cliente/usuario?
    - ¿Validáis con usuarios al ritmo que producís?
4. Respuesta al cambio vs. seguimiento del plan:
    - ¿Podéis cambiar de rumbo fácilmente cuando la evidencia lo requiere?
    - ¿La IA os da más o menos capacidad de detectar cambios en el entorno?

Para cada valor, puntúa del 1 al 10 dónde está tu equipo y qué podría mejorar.

Ejercicio 2: mapa de amplificación

La IA amplifica lo que el equipo ya trae. Identifica qué está amplificando en tu caso:

Fortalezas que la IA está amplificando: (cosas que hacíais bien y ahora hacéis mejor gracias a la IA)

1.

2.

3.

Debilidades que la IA está exponiendo: (problemas que teníais y ahora son más visibles o graves)

1.

2.

3.

Para cada debilidad expuesta, ¿qué acción podría mitigarla? Recuerda: la solución no es técnica (más IA), sino humana (mejores prácticas, más reflexión, más validación).

Ejercicio 3: diagnóstico velocidad vs. agilidad

Responde estas preguntas para tu equipo:

VELOCIDAD:

- ¿Cuántas features/historias entregáis por sprint ahora vs. hace un año?
- ¿Cuánto tiempo pasa desde que se identifica una necesidad hasta que hay código en producción?

AGILIDAD:

- ¿Cuánto tiempo pasa desde que hay código en producción hasta que sabéis si funciona para los usuarios?
- ¿Cuándo fue la última vez que cambiasteis significativamente el rumbo del producto basándoos en feedback?
- ¿Cuándo fue la última vez que la retrospectiva produjo un cambio real en vuestra forma de trabajar?

CALIDAD:

- ¿Ha aumentado o disminuido la deuda técnica en el último año?
- ¿Los bugs en producción han aumentado o disminuido?
- ¿La confianza del equipo en el código que produce ha aumentado o disminuido?

Si la velocidad ha aumentado pero la agilidad o la calidad han disminuido, tenéis un problema que no se resuelve con más velocidad.

Resumen del capítulo

En este capítulo hemos establecido los principios que deben guiar la evolución de scrum ante la IA:

- **Los cuatro valores del *Manifiesto Ágil*** (individuos sobre procesos, software funcionando, colaboración con el cliente, respuesta al cambio) no solo sobreviven a la IA: **se amplifican.**
- **La IA es un amplificador de lo que el equipo ya trae.** Potencia las fortalezas y expone las debilidades. No es neutral.
- **Velocidad y agilidad no son lo mismo.** La agilidad es la capacidad de responder al cambio mientras se entrega valor. La velocidad sin agilidad produce más código, pero no más valor.
- **Lo que no debe cambiar:** empirismo (transparencia, inspección, adaptación), reflexión deliberada, colaboración humana y autoorganización.

En el próximo capítulo exploraremos cómo estos principios se traducen en la transformación de los roles: product owner, desarrolladores y scrum master ante la era de la IA.

Preguntas de autoevaluación

1. ¿Cómo se amplifica el valor 'individuos e interacciones sobre procesos y herramientas' con la IA?
2. ¿Qué significa que la IA es un 'amplificador' de lo que el equipo ya trae?
3. ¿Cuál es la diferencia entre velocidad y agilidad? Pon un ejemplo.
4. ¿Por qué el empirismo se vuelve más importante, no menos, con la IA?
5. ¿Qué riesgo corre un equipo que elimina las retrospectivas para 'ir más rápido' con la IA?

CAPÍTULO 3 — Roles en transformación

Los roles habituales en equipos scrum —product owner, scrum master, developers— no desaparecen con la IA, pero experimentan una transformación profunda en su foco, sus competencias y su relevancia relativa. En este capítulo exploraremos cómo evoluciona cada rol, qué nuevas competencias se vuelven críticas, y cómo la pregunta sobre si la IA es un «miembro del equipo» afecta a la dinámica de trabajo.

3.1. Product owner → AI architect

Con la IA capaz de construir features más rápido de lo que la organización puede validar su deseabilidad, el product owner se convierte en el factor limitante de la entrega de valor. Paradójicamente, esto hace que su rol sea más importante, no menos.

El nuevo cuello de botella

Durante décadas, el equipo de desarrollo fue el recurso escaso. El product owner priorizaba el backlog sabiendo que la capacidad de construcción era limitada. Ahora, la capacidad de construcción se ha multiplicado exponencialmente. La pregunta ya no es «¿qué podemos construir?» sino «¿qué deberíamos construir?».

Este cambio desplaza la responsabilidad crítica hacia la gestión de producto. Un product owner que se limita a escribir historias de usuario y priorizar un backlog se queda corto. Se necesita un perfil que pueda:

- Gestionar activamente qué merece ser construido.
- Diseñar experimentos de validación continua.
- Decidir dónde la IA aporta valor real frente a dónde genera ruido.
- Mantener la visión estratégica cuando la velocidad invita a perderse en los detalles.

Las competencias emergentes incluyen:

- **Data literacy:** capacidad de interpretar datos, diseñar métricas de éxito y tomar decisiones basadas en evidencia. Con la IA generando más outputs, la capacidad de medir su impacto real es crítica.
- **Estrategia de producto con IA:** entender qué puede y qué no puede hacer la IA. Saber cuándo un problema se resuelve mejor con IA y cuándo con otras aproximaciones. Anticipar cómo la IA cambiará el mercado y los competidores.
- **Gestión de la paradoja producción-validación:** diseñar flujos de trabajo donde la capacidad de validar escale con la capacidad de producir. Si el equipo puede generar diez features por sprint, ¿cómo validamos diez features por sprint?

- **Comunicación con equipos técnicos sobre IA:** sin necesidad de ser técnico, entender lo suficiente para tomar decisiones informadas sobre qué tareas delegar a la IA, qué nivel de revisión exigir y qué riesgos asumir.

El riesgo de la obsolescencia

El riesgo de obsolescencia para este rol es bajo si evoluciona hacia la visión estratégica. Es alto si se limita a escribir historias de usuario, una tarea que la IA puede asistir significativamente.

Un product owner que usa la IA para generar borradores de historias, analizar feedback de usuarios y explorar opciones de producto libera tiempo para lo que la IA no puede hacer: entender el negocio, construir relaciones con stakeholders y tomar decisiones estratégicas.

3.2. Desarrolladores → Product builders

El equipo de desarrollo se transforma en lo que diversos autores denominan «product builders»: perfiles que combinan el uso de herramientas no-code, la generación de código asistida por IA y una comprensión profunda del producto.

El desplazamiento del valor

El valor de un desarrollador se desplaza desde la implementación hacia la orquestación:

- **Antes:** el valor estaba en saber escribir código eficiente, conocer el lenguaje, dominar el framework.
- **Ahora:** el valor está en saber qué pedir a la IA, cómo evaluar lo que genera, cuándo rechazarlo y cómo integrarlo en una arquitectura coherente.

Esto no significa que las competencias técnicas sean irrelevantes. Al contrario: se necesita más conocimiento técnico para evaluar código generado por IA que para escribirlo desde cero. Un desarrollador junior puede aceptar código de la IA que un senior identificaría como problemático.

Nuevas competencias esenciales

- **Prompt engineering:** la capacidad de comunicarse efectivamente con la IA. No es solo una habilidad técnica; es una forma de pensar el problema de manera que la IA pueda procesarlo.
- **Spec writing:** la escritura de especificaciones claras, precisas y testeables que guíen a la IA. Una spec mal escrita produce código mal generado.

- **Revisión de código generado:** saber qué buscar en código que no has escrito. Detectar patrones problemáticos, duplicaciones ocultas, vulnerabilidades de seguridad. Esto requiere más experiencia, no menos.
- **Context engineering:** proporcionar a la IA el contexto adecuado para obtener resultados útiles. Esto incluye la arquitectura existente, los estándares del equipo, el historial de decisiones técnicas.

El dato contraintuitivo

Un dato revelador de la investigación: desarrolladores senior experimentaron una disminución del 19% en su rendimiento en tareas complejas al usar IA, debido al tiempo dedicado a revisar y probar el output generado.

Esto no significa que los seniors no deban usar IA. Significa que la IA no es un atajo mágico: requiere inversión de tiempo y juicio experto. Los seniors aportan ese juicio; los juniors pueden no tenerlo aún.

Tamaño del equipo

La recomendación clásica de 3-9 desarrolladores está en revisión. Equipos de 2-3 personas con asistencia de IA pueden lograr lo que antes requería equipos mayores. La investigación de Harvard encontró que individuos con IA igualaron el rendimiento de equipos enteros sin IA.

Esto tiene **implicaciones** para la organización del trabajo:

- Menos necesidad de coordinación compleja.
- Más autonomía por persona.
- Roles más fluidos y menos especializados.
- Eventos potencialmente más ligeros.

Pero también **riesgos:**

- Menos diversidad de perspectivas.
- Mayor dependencia de individuos clave.
- Menos resiliencia ante bajas.
- Riesgo de silos de conocimiento.

3.3. Scrum master: transformación o extinción

Este es el rol con mayor riesgo de obsolescencia si su valor se define exclusivamente como «facilitador de eventos». Pero también es el rol que más puede ganar si evoluciona hacia competencias que la IA no puede replicar.

El riesgo real

La IA puede hacer muchas de las tareas operativas del scrum master tradicional:

- Sugerir técnicas de facilitación para cada tipo de reunión.
- Generar agendas basadas en el contexto del equipo.
- Tomar notas y resúmenes de reuniones.
- Analizar métricas de flujo e identificar cuellos de botella.
- Detectar patrones en retrospectivas pasadas.
- Proponer acciones de mejora basadas en datos.

Si el valor del scrum master se limita a estas tareas, la IA lo hace prescindible.

Lo que la IA no puede hacer

Sin embargo, las competencias que la IA no puede replicar son precisamente las que definen el valor de un scrum master evolucionado:

- **Inteligencia emocional:** leer el estado emocional del equipo, detectar conflictos latentes, crear seguridad psicológica. La IA puede analizar el sentimiento de un texto; no puede sentir la tensión en una sala.
- **Pensamiento sistémico:** ver las conexiones entre problemas aparentemente aislados, entender cómo un cambio en un lugar afecta al sistema completo. La IA puede procesar datos; el pensamiento sistémico requiere visión.
- **Liderazgo de equipo**: inspirar, motivar, dar feedback difícil con empatía, sostener al equipo en momentos de crisis. Estas son competencias profundamente humanas.
- **Toma de decisiones éticas:** decidir cuándo una práctica es aceptable y cuándo cruza una línea. La ética requiere juicio humano.
- **Coaching organizacional:** ayudar a la organización a transformarse, no solo al equipo. Navegar la política organizacional, influir sin autoridad, construir alianzas.

La evolución del rol

Las proyecciones para 2030 sitúan este rol como un líder centrado en estrategia, transformación cultural y coaching organizacional, alejado de la operativa diaria del equipo.

El scrum master evolucionado:

- Facilita sistemas de trabajo híbridos humano-IA.
- Establece la gobernanza del uso de IA en el equipo.
- Coaching sobre las nuevas competencias (prompt engineering, revisión de código generado).
- Lidera la reflexión sobre cómo la IA está cambiando la forma de trabajar.

- Protege los momentos de reflexión contra la presión de la velocidad.

En equipos maduros y pequeños

En equipos pequeños (2-3 personas) y maduros, es posible que las responsabilidades del scrum master estén completamente interiorizadas por el equipo, sin necesidad de un rol dedicado. Esto no es nuevo —siempre fue posible en equipos muy maduros—, pero la IA lo hace más frecuente.

El scrum master puede entonces evolucionar hacia un rol de servicio a múltiples equipos, o hacia funciones de transformación organizacional.

3.4. La IA: ¿comprometida o implicada?

Una pregunta emergente que afecta a todos los roles es dónde sitúa el equipo a la IA en su dinámica de trabajo. ¿Es un miembro más del equipo? ¿Una herramienta sofisticada? ¿Un colaborador externo?

La distinción clásica

En scrum se distingue entre «comprometidos» (quienes intervienen directamente en la construcción del producto) e «implicados» (otras partes interesadas). Los comprometidos son responsables del resultado; los implicados aportan pero no responden.

¿Dónde encaja la IA en esta distinción?

La postura del *Scrum Guide Expansion Pack*

El *Scrum Guide Expansion Pack* (2025) de Sutherland, Coleman y Jocham sugiere que la IA puede ser un «product developer» más, con la condición de que al menos un humano mantenga la responsabilidad. Renombra «developers» a «product developers» y explicita que estos «pueden ser humanos o automatizados».

Esta postura trata a la IA como un comprometido potencial, integrado en el equipo.

La postura alternativa

Otras voces de la comunidad consideran que tratar a la IA como «miembro del equipo» es una analogía útil pero imperfecta:

- La IA no tiene compromiso: No se siente responsable del resultado. No sufre cuando el proyecto fracasa.
- La IA no tiene motivación: No se esfuerza más cuando el deadline aprieta. No se desmotiva cuando el trabajo es tedioso.

- La IA no participa en retrospectivas de manera significativa: No reflexiona sobre su proceso ni propone mejoras.
- La IA no desarrolla relaciones de confianza: No construye rapport con los compañeros ni con los stakeholders.

Desde esta perspectiva, la IA es una herramienta potentísima al servicio de los comprometidos, pero no es en sí misma un comprometido.

Implicaciones prácticas

Independientemente de la postura filosófica, hay implicaciones prácticas que cada equipo debe decidir:

- ¿Quién rinde cuentas cuando un agente de IA comete un error? La responsabilidad siempre debe recaer en un humano. Si el código generado por IA tiene un bug, ¿quién responde? Debe haber claridad.
- ¿Cómo se «onboardea» a una IA al equipo? La propuesta del AI teammate Framework de Scrum.org incluye «Context Management»: proporcionar a la IA el contexto necesario (estándares, arquitectura, historial) para que sus outputs sean relevantes. Este onboarding es diferente al de un humano, pero es necesario.
- ¿Necesita la IA participar en los eventos? No de manera literal. Pero los eventos deben contemplar la IA: en la planning se decide qué tareas se delegan a la IA; en la daily se revisa el output de la IA; en la retrospectiva se evalúa cómo funciona la colaboración con la IA.

La recomendación práctica

Más allá del debate conceptual, lo importante es que cada equipo defina explícitamente:

- Qué puede y qué no puede hacer la IA en este equipo.
- Quién es responsable de revisar lo que la IA produce.
- Cómo se integra la IA en el flujo de trabajo diario.
- Qué límites ponemos al uso de la IA.

Estas decisiones son parte de la autoorganización del equipo.

Ejercicios prácticos

Ejercicio 1: análisis de evolución de tu rol

Reflexiona sobre cómo la IA está afectando a tu rol específico:

Si eres product owner / product manager:

- ¿Qué proporción de tu tiempo dedicas a escribir historias vs. a validar con usuarios?
- ¿Has incorporado la IA para generar borradores de historias, analizar feedback o explorar opciones?
- ¿Cuál es tu mayor desafío: generar ideas o validarlas?
- ¿Qué competencia necesitas desarrollar más: data literacy, estrategia de IA, comunicación con técnicos?

Si eres desarrollador:

- ¿Qué proporción de tu código es generado/asistido por IA?
- ¿Cuánto tiempo dedicas a revisar código generado vs. a escribir código nuevo?
- ¿Has notado cambios en la calidad del código desde que usas IA?
- ¿Qué competencias necesitas desarrollar más: prompt engineering, spec writing, revisión de código, context engineering?

Si eres scrum master / agile coach:

- ¿Qué proporción de tu tiempo dedicas a facilitar eventos vs. a coaching/transformación?
- ¿Has incorporado la IA para preparar retrospectivas, analizar métricas o generar informes?
- ¿Tu equipo te necesita para la operativa diaria o funcionaría sin ti?
- ¿Qué competencia necesitas desarrollar más: sistemas híbridos, gobernanza de IA, coaching organizacional?

Ejercicio 2: mapa de responsabilidades con IA

Crea un mapa de responsabilidades que incluya la IA:

Para cada tipo de tarea habitual en tu equipo, define:

Tarea	¿La hace humano, IA, o ambos?	Si ambos, ¿quién hace qué?	¿Quién es responsable del resultado?

Ejemplos de tareas:

- Escribir historias de usuario.
- Diseñar la arquitectura.

- Escribir código nuevo.
- Revisar código.
- Escribir tests.
- Analizar métricas.
- Preparar retrospectivas.
- Documentar decisiones.

¿Hay tareas donde no está clara la responsabilidad? ¿Hay tareas que nadie revisa cuando las hace la IA? Esas son áreas de riesgo.

Ejercicio 3: plan de desarrollo de competencias

Basándote en la evolución de los roles, identifica las competencias que necesitas desarrollar:

1. Lista las 3 competencias nuevas más relevantes para tu rol en la era de la IA:

-
-
-

2. Para cada una, evalúa tu nivel actual (1-10):

3. Para la competencia donde tienes más brecha, diseña un plan de desarrollo:

- ¿Qué recursos de aprendizaje puedes usar? (cursos, libros, mentores)
- ¿Qué práctica deliberada puedes hacer? (proyectos, ejercicios)
- ¿Cómo medirás tu progreso?
- ¿En qué plazo esperas mejorar significativamente?

4. ¿Qué apoyo necesitas de tu organización para desarrollar estas competencias?

Resumen del capítulo

En este capítulo hemos explorado la transformación de los roles en equipos scrum:

- El product owner evoluciona hacia AI architect: el nuevo cuello de botella estratégico. Su valor está en decidir qué construir y validar que lo construido genera valor. Nuevas competencias: data literacy, estrategia de producto con IA, gestión de la paradoja producción-validación.
- Los desarrolladores evolucionan hacia product builders: su valor se desplaza de la implementación a la orquestación. Nuevas competencias: prompt engineering, spec writing, revisión de código generado, context engineering.
- El scrum master enfrenta transformación o extinción: si su valor se limita a facilitar eventos, la IA lo hace prescindible. Su futuro está en inteligencia emocional, pensamiento sistémico, liderazgo y coaching organizacional.

- La IA plantea la pregunta de si es un «comprometido» o una herramienta. Independientemente de la respuesta conceptual, cada equipo debe definir explícitamente qué puede hacer la IA, quién revisa lo que produce y quién es responsable del resultado.

En el próximo capítulo exploraremos cómo evolucionan los artefactos: de historias de usuario a especificaciones, y la Definition of Done para código generado por IA.

Preguntas de autoevaluación

1. ¿Por qué el product owner se convierte en el 'nuevo cuello de botella' con la IA?
2. ¿Qué significa que el valor del desarrollador se 'desplaza de la implementación a la orquestación'?
3. ¿Cuáles son las competencias que la IA no puede replicar y que definen el valor del scrum master evolucionado?
4. ¿Qué postura adopta el *Scrum Guide Expansion Pack* sobre la IA como miembro del equipo?
5. ¿Qué debe definir explícitamente cada equipo sobre el uso de la IA, independientemente del debate conceptual?

CAPÍTULO 4 — Artefactos adaptados

Los artefactos de scrum —pila del producto, pila del sprint, incremento— necesitan evolucionar para contemplar la IA. No se trata de reemplazarlos, sino de adaptarlos a una realidad donde el lenguaje de comunicación incluye a la máquina y donde la calidad del código generado requiere nuevos controles.

En este capítulo exploraremos la tensión entre historias de usuario y especificaciones, el emergente Spec-Driven Development, y cómo adaptar la Definition of Done al código generado por IA.

4.1. De historias de usuario a especificaciones

Las historias de usuario nacieron como herramienta de comunicación entre humanos: «Como [usuario], quiero [acción] para [beneficio]». Este formato es eficaz para transmitir intención entre personas, pero no es la forma más precisa de instruir a una IA generativa.

La tensión fundamental

Surge una tensión entre la comunicación humana y la instrucción a la máquina:

- **Para humanos:** las historias de usuario funcionan bien. Son breves, capturan la intención, dejan espacio para la conversación. Un desarrollador humano puede hacer preguntas, inferir contexto, proponer alternativas.
- **Para IA:** las historias de usuario son ambiguas. La IA necesita instrucciones más precisas. No infiere contexto que no se le proporciona explícitamente. Interpreta literalmente lo que se le dice.

Esto no significa que las historias de usuario deban desaparecer. Significa que necesitan convivir con nuevos formatos.

El formato de la especificación

Una especificación para IA incluye típicamente:

- **Contexto:** qué existe ya, cómo encaja esta funcionalidad en el sistema, qué restricciones aplican.
- **Requisitos funcionales:** qué debe hacer exactamente, con casos de uso específicos, inputs esperados, outputs esperados.
- **Requisitos no funcionales:** rendimiento, seguridad, accesibilidad, compatibilidad.
- **Criterios de aceptación:** cómo sabemos que está bien hecho, idealmente con tests automatizables.
- **Ejemplos:** casos concretos que ilustran el comportamiento esperado.

Este formato es más largo que una historia de usuario, pero permite a la IA generar código más alineado con la intención.

La convivencia, no la sustitución

La solución no es elegir entre historias de usuario y specs. Es encontrar la convivencia adecuada:

- **No todo necesita el mismo nivel de especificación:** un prototipo de validación puede partir de una historia de usuario. Una funcionalidad crítica para producción requiere una spec detallada. El nivel de formalización depende del riesgo.
- **La spec complementa, no sustituye, la conversación:** el formato «Conversation, Card, Confirmation» sigue siendo válido. La spec traduce esa conversación a un lenguaje procesable por la IA, pero la conversación sigue siendo necesaria para alinear intenciones.
- **Specs como documentos vivos:** las especificaciones deben iterar con el mismo espíritu que un backlog ágil. Se refinan, se adaptan y evolucionan con cada ciclo de aprendizaje. No son documentos estáticos que se escriben una vez y se ejecutan.

El riesgo de regresión waterfall

Si las especificaciones se vuelven demasiado rígidas, se corre el peligro de regresar al modelo de cascada. Programadores experimentados advierten que especificaciones excesivamente formalizadas pueden causar problemas innecesarios y ralentizar los ciclos de cambio y retroalimentación.

La clave es mantener el espíritu iterativo: la spec no es un contrato inamovible sino una hipótesis que se refina con el feedback del código generado.

4.2. Spec-Driven Development en profundidad

El Spec-Driven Development (SDD) es una de las prácticas más significativas que han emergido en 2025. Es un paradigma que utiliza especificaciones de requisitos bien elaboradas como prompts para que los agentes de codificación IA generen código ejecutable.

El concepto central

La especificación se convierte en la «fuente de verdad» tanto para el humano como para la IA. El código es un derivado de la spec, no un artefacto independiente. Mantener el software significa, en última instancia, evolucionar las especificaciones.

A diferencia de las especificaciones tradicionales (leídas por humanos y frecuentemente ignoradas), las specs del SDD son artefactos ejecutables: se ejecutan durante la validación, y la implementación no puede desviarse sin provocar fallos.

Las cuatro fases del SDD

Herramientas como GitHub Spec Kit han formalizado este flujo:

- **Specify:** se describe a alto nivel qué se quiere construir y por qué. El agente genera una especificación detallada basándose en la descripción.
- **Plan:** el agente propone un plan de implementación basado en la spec. Este plan incluye la secuencia de pasos, las dependencias y los puntos de decisión.
- **Tasks:** la spec y el plan se descomponen en tareas pequeñas, revisables e implementables de forma aislada. Cada tarea es lo suficientemente acotada para ser verificable.
- **Implement:** el agente ejecuta las tareas y el desarrollador revisa cambios focalizados, no bloques masivos de código. La revisión es más efectiva porque cada cambio está acotado.

Niveles de madurez

No todos los equipos adoptan SDD al mismo nivel:

- **Nivel 1 - Spec-first development:** los equipos escriben especificaciones antes de codificar para guiar la implementación asistida por IA. Es el nivel de entrada.
- **Nivel 2 - Spec-driven con validación automatizada:** las specs incluyen escenarios BDD (Behavior-Driven Development), tests de contrato API o simulaciones que se ejecutan automáticamente. La spec no solo guía sino que verifica.
- **Nivel 3 - Spec-as-source-of-truth:** la especificación reemplaza al código como artefacto principal. El código es generado y regenerado desde la spec. Mantener el software significa mantener las specs.

Estado de madurez

Es importante ser realista: el SDD es una práctica emergente a finales de 2025. Herramientas como GitHub Spec Kit, Kiro y Tessl están en fases tempranas. Los agentes de IA aún no siguen todas las instrucciones de las specs de manera fiable.

Adoptar SDD requiere:

- Experimentación (probar en proyectos de bajo riesgo primero).
- Paciencia (los resultados no son inmediatos).
- Iteración (la práctica de escribir buenas specs se desarrolla con el tiempo).
- Complemento con revisión humana (no confiar ciegamente en el output).

4.3. La pila del producto con specs

La pila del producto sigue siendo el inventario de funcionalidades que deben incorporarse al producto. Pero su contenido y formato evolucionan para contemplar la IA.

Una pila híbrida

En la práctica, la pila del producto contendrá diferentes tipos de elementos:

- **Historias de usuario tradicionales:** para funcionalidades simples, prototipado rápido, o fases tempranas de exploración donde la flexibilidad es más importante que la precisión.
- **Especificaciones detalladas:** para funcionalidades complejas, código de producción, o cualquier trabajo donde la IA necesite instrucciones precisas.
- **Épicas con specs asociadas:** las épicas se descomponen no solo en historias sino también en specs para las partes que requieren más detalle.

La clave es que el formato se adapte a la necesidad, no al revés.

Refinamiento evolucionado

La actividad de refinamiento (grooming) del backlog adquiere una dimensión nueva:

- **Antes:** el refinamiento consistía en detallar historias, estimar esfuerzo y asegurar que estaban «ready» para entrar en un sprint.
- **Ahora:** el refinamiento incluye decidir qué nivel de especificación necesita cada elemento. ¿Basta una historia? ¿Necesita una spec completa? ¿Qué contexto debe incluir la spec para que la IA genere código útil?

Las preguntas del refinamiento evolucionan:

- ¿Qué necesita saber la IA para implementar esto correctamente?
- ¿Qué criterios de aceptación podemos automatizar?
- ¿Qué ejemplos concretos ilustran el comportamiento esperado?
- ¿Qué restricciones del sistema existente debe conocer la IA?

Definition of Ready para Specs

Si el equipo trabaja con SDD, la Definition of Ready se amplía:

Una spec está ready cuando:

- Tiene contexto suficiente (arquitectura, restricciones, dependencias).
- Los requisitos funcionales son precisos y sin ambigüedad.
- Los criterios de aceptación son verificables (idealmente automatizables).
- Incluye ejemplos concretos de comportamiento esperado.

- Ha sido revisada por al menos un miembro del equipo.
- Tiene un tamaño que permite revisión efectiva del output.

Esta DoR es más exigente que la tradicional, pero el coste de escribir una buena spec se compensa con la calidad del código generado.

La pila como radiador de información

La pila del producto sigue siendo un radiador de información mediante el que el propietario del producto y el equipo comparten la visión. Pero ahora también es un repositorio de contexto para la IA.

Algunas **prácticas emergentes:**

- Vincular las specs con la documentación de arquitectura.
- Mantener un «contexto base» que se incluye en todas las specs.
- Versionar las specs junto con el código.
- Usar formatos que la IA pueda procesar directamente.

4.4. Definition of Done para código generado por IA

La Definition of Done (DoD) necesita adaptarse específicamente cuando parte del código es generado por IA. Los datos de la industria muestran que, sin controles deliberados, la IA acelera la acumulación de deuda técnica.

Los datos preocupantes

Análisis de grandes bases de código muestran patrones que requieren atención:

- La duplicación de código en proyectos asistidos por IA aumentó significativamente (del ~8% al ~12% de las líneas cambiadas)
- La actividad de refactoring cayó drásticamente: los equipos producen código nuevo en lugar de mejorar el existente
- En 2024, por primera vez, el código copiado o generado por IA superó al código refactorizado
- La confianza de los desarrolladores en la precisión del código generado por IA cayó del 69% al 54%

Estos datos sugieren que la velocidad puede estar erosionando la calidad. La DoD es el mecanismo para contrarrestarlo.

DoD ampliada para el código de IA

Una Definition of Done adaptada incluye criterios específicos:

1. **Revisión humana obligatoria:**

    - Todo código generado por IA ha sido revisado por al menos un desarrollador.
    - El revisor entiende lo que hace el código (no solo que «funciona»).
    - El revisor ha verificado que el código cumple los estándares del equipo.
2. **Análisis de calidad:**
    - El código pasa los análisis estáticos configurados (linting, formateo).
    - No se ha aumentado la duplicación de código por encima del umbral definido.
    - La complejidad ciclomática está dentro de los límites aceptados.
    - No hay vulnerabilidades de seguridad detectadas.
3. **Testing:**
    - Los tests unitarios cubren la nueva funcionalidad.
    - Los tests de integración pasan.
    - Los tests de aceptación definidos en la spec pasan.
4. **Documentación:**
    - El código está documentado según los estándares del equipo.
    - Los cambios están reflejados en la documentación de arquitectura si aplica.

Métricas de salud del código

Además de la DoD, los equipos están incorporando métricas continuas de salud del código como contrapeso a la velocidad:

- **Ratio de código generado vs. código refactorizado:** ¿Estamos solo añadiendo o también mejorando?
- **Tendencia de duplicación:** ¿La duplicación aumenta o se mantiene estable?
- **Cobertura de tests del código generado:** ¿Testeamos lo que genera la IA al mismo nivel que lo que escribimos?
- **Tiempo de revisión por línea generada:** ¿Dedicamos tiempo suficiente a revisar?

Estas métricas se revisan en retrospectivas para detectar tendencias preocupantes.

IA supervisando IA

Un modelo emergente es utilizar una IA como revisor automático de lo que otra IA genera:

- Análisis automático de calidad antes de la revisión humana.
- Detección de patrones problemáticos conocidos.
- Verificación de cumplimiento de estándares.
- «Semáforo» que indica el nivel de confianza en el código generado.

Este modelo no sustituye la revisión humana, pero la hace más eficiente al filtrar los problemas obvios.

Ejercicios prácticos

Ejercicio 1: convertir una historia en spec

Toma una historia de usuario de tu backlog actual y conviértela en una especificación para IA:

Historia de usuario original

```
Como [usuario], quiero [funcionalidad] para [beneficio]
```

Ahora amplíala:

```
CONTEXTO
   • ¿Cómo encaja en el sistema existente?
   • ¿Qué componentes/servicios están involucrados?
   • ¿Qué restricciones aplican?

REQUISITOS FUNCIONALES
   • ¿Qué debe hacer exactamente?
   • ¿Cuáles son los inputs esperados?
   • ¿Cuáles son los outputs esperados?
   • ¿Qué casos edge existen?

REQUISITOS NO FUNCIONALES
   • ¿Qué rendimiento se espera?
   • ¿Qué consideraciones de seguridad aplican?

CRITERIOS DE ACEPTACIÓN
   • ¿Cómo sabemos que está bien hecho?
   • ¿Qué tests podemos automatizar?

EJEMPLOS
   • Describe 2-3 casos concretos de uso
```

¿Cuánto más larga es la spec que la historia? ¿Qué información nueva incluye?

Ejercicio 2: auditoría de tu Definition of Done

Revisa la Definition of Done actual de tu equipo y evalúa si contempla el código generado por IA:

DoD actual del equipo

(lista los criterios actuales)

Para cada criterio, evalúa:

- ☐ ¿Aplica igual al código generado por IA?

- ☐ ¿Necesita adaptación?
- ☐ ¿Falta algo específico para código de IA?

Criterios adicionales que podrían ser necesarios:

- ☐ Revisión humana obligatoria
- ☐ Verificación de no-duplicación
- ☐ Análisis de seguridad específico
- ☐ Comprensión del código (no solo que "funciona")
- ☐ Otros: _____

Propuesta de DoD ampliada:

(escribe la nueva versión)

¿Cómo introducirías estos cambios en tu equipo?

Ejercicio 3: métricas de salud del código

Define las métricas de salud del código que tu equipo debería monitorizar:

1. **Métricas de calidad:**
   - Duplicación de código: ¿cuál es el umbral aceptable? ____%
   - Complejidad ciclomática máxima: _____
   - Cobertura de tests mínima: ____%
2. **Métricas de proceso:**
   - Ratio código generado vs. refactorizado: ¿cuál es el objetivo? _____
   - Tiempo medio de revisión por PR con código de IA: _____
   - % de PRs que requieren cambios después de la primera revisión: _____
3. **Métricas de resultado:**
   - Bugs en producción relacionados con código de IA: _____
   - Incidentes de seguridad relacionados con código de IA: _____
4. ¿Cómo recopilaréis estas métricas?
5. ¿Con qué frecuencia las revisaréis?
6. ¿Qué haréis si detectáis una tendencia negativa?

Resumen del capítulo

En este capítulo hemos explorado cómo evolucionan los artefactos de scrum:

- Las historias de usuario necesitan convivir con especificaciones más detalladas para instruir a la IA. No es sustitución sino convivencia: el nivel de formalización depende del riesgo.

- El Spec-Driven Development (SDD) utiliza especificaciones como fuente de verdad. Sus cuatro fases (Specify, Plan, Tasks, Implement) proporcionan un flujo estructurado. Es una práctica emergente que requiere experimentación.
- La pila del producto se convierte en una pila híbrida con historias y specs. El refinamiento incluye decidir qué nivel de especificación necesita cada elemento.
- La Definition of Done necesita criterios específicos para código generado por IA: revisión humana obligatoria, análisis de calidad, verificación de no-duplicación, y métricas de salud del código.

En el próximo capítulo exploraremos cómo evolucionan los eventos de scrum: sprints, dailies, reviews y retrospectivas en la era de la IA.

Preguntas de autoevaluación

1. ¿Por qué las historias de usuario son 'ambiguas' para la IA mientras que funcionan bien para humanos?
2. ¿Cuáles son las cuatro fases del Spec-Driven Development según GitHub Spec Kit?
3. ¿Qué debe incluir una Definition of Ready específica para especificaciones?
4. ¿Qué datos de la industria justifican la necesidad de adaptar la Definition of Done para código de IA?
5. ¿Qué significa 'IA supervisando IA' en el contexto del QA?

CAPÍTULO 5 — Eventos evolucionados

Los eventos de scrum —sprint, planning, daily, review, retrospectiva— son los mecanismos que hacen operativos los principios de empirismo y mejora continua. Con la IA, estos eventos no desaparecen, pero su formato, duración y contenido necesitan adaptarse. En este capítulo exploraremos cómo evoluciona cada evento, qué tensiones introduce la velocidad de la IA, y por qué la retrospectiva se vuelve más necesaria precisamente cuando la presión invita a eliminarla.

5.1. La cadencia del sprint en la era de la IA

El sprint de dos semanas se ha convertido en el estándar de facto en la mayoría de equipos scrum. Pero conviene recordar que esta duración no siempre fue la norma: las primeras iteraciones de scrum en los años 90 utilizaban ciclos de uno o dos meses, y fue la propia comunidad la que fue acortando la cadencia a medida que el contexto lo requería.

Con la IA, esta cadencia entra de nuevo en tensión: los equipos pueden generar incrementos de producto, MVPs o funcionalidades en cuestión de horas o días.

Las opciones que los equipos están explorando

Opción 1: mantener sprints con micro-iteraciones internas

Conservar el sprint como contenedor de planificación y revisión, pero dentro de él ejecutar múltiples ciclos cortos de ideación-generación-validación asistidos por IA.

- **Ventajas:** preserva los puntos de sincronización. Mantiene el ritmo de reflexión. Compatible con stakeholders que esperan ciclos regulares.
- **Riesgos:** puede sentirse artificial si la velocidad real es mucho mayor.

Opción 2: Sprints más cortos

Reducir a una semana o menos, acercando la cadencia a la velocidad real de entrega. Algunos equipos experimentan con sprints de 3 días para trabajo asistido por IA.

- **Ventajas:** mayor frecuencia de revisión y adaptación. Feedback más rápido.
- **Riesgos:** overhead de planning/review puede consumir proporción excesiva del tiempo. Puede ser demasiado frecuente para stakeholders.

Opción 3: Flujo continuo (Kanban-style)

Abandonar el sprint como contenedor temporal y adoptar un flujo continuo con límites de trabajo en curso (WIP).

- **Ventajas:** elimina la artificialidad del timeboxing. Maximiza la flexibilidad.
- **Riesgos:** pierde los momentos de reflexión estructurada que el sprint proporciona. Requiere madurez para mantener la disciplina de revisión.

Opción 4: Cadencia dual

Un ritmo rápido para prototipado y experimentación (días) y un ritmo más pausado para ingeniería de producción y validación estratégica (semanas).

- **Ventajas:** adapta la cadencia al tipo de trabajo. Compatible con el modelo de doble carril.
- **Riesgos:** complejidad de gestionar dos ritmos. Puede ser confuso para el equipo.

El riesgo de eliminar la cadencia

Eliminar los sprints por completo conlleva un riesgo real: perder los momentos de reflexión estructurada. La retrospectiva, por ejemplo, es precisamente el mecanismo que permite al equipo inspeccionar y adaptar su forma de trabajar con IA.

Paradójicamente, es ahora cuando más se necesita reflexionar sobre el proceso, justo cuando la velocidad invita a omitir estos momentos.

La pregunta para tu equipo

¿Nuestro sprint actual es un contenedor útil que nos da ritmo y momentos de reflexión, o es un cuello de botella que frena nuestra capacidad de entrega real? La respuesta depende del contexto. Lo importante es que sea una decisión consciente, no inercia.

5.2. Planning y daily evolucionados

Los eventos dentro del sprint también evolucionan para contemplar la IA.

La planning cuando la IA cambia la ecuación

La reunión de planificación del sprint sigue dando respuesta a tres cuestiones: ¿por qué es valioso este sprint? ¿qué se puede hacer? ¿cómo se va a hacer? Pero el contenido de estas cuestiones cambia.

¿Qué se puede hacer? Con la IA, el equipo puede hacer mucho más de lo que puede validar. La planning debe incorporar explícitamente la validación como parte del trabajo del sprint, no solo la construcción.

Pregunta nueva: «Para cada elemento que queremos construir, ¿cómo lo validaremos dentro del sprint?»

¿Cómo se va a hacer? Si el equipo trabaja con Spec-Driven Development, parte de la planificación se desplaza hacia la escritura o refinamiento de especificaciones que guiarán a la IA.

Decisiones nuevas:

- ¿Qué tareas se delegan a la IA?
- ¿Cuáles requieren trabajo humano exclusivo?
- ¿Qué mecanismos de revisión se aplicarán al output generado?
- ¿Qué nivel de especificación necesita cada elemento?

Estimación: ¿qué estamos estimando?

La IA introduce una complejidad nueva en la estimación. Una historia que antes costaba 8 puntos puede costar 2 de generación con IA, pero 5 de revisión humana. La relación entre esfuerzo y resultado ya no es estable.

Opciones que los equipos están explorando:

- Separar la estimación de generación y la estimación de revisión.
- Usar métricas de flujo (cycle time, lead time) en lugar de puntos.
- Adoptar #NoEstimates en equipos maduros con flujo continuo.
- Mantener puntos pero recalibrar su significado.

El scrum diario en equipos reducidos

En equipos de 2-3 personas con asistencia de IA, la daily en su formato clásico puede resultar excesiva. Las opciones incluyen:

- **Check-in asíncrono:** un mensaje en el canal del equipo al inicio del día. Suficiente para equipos pequeños y maduros.
- **Frecuencia reducida:** 2-3 veces por semana en lugar de diario.
- **Foco en la IA:** centrar la daily en revisar el output de la IA del día anterior y decidir qué necesita revisión humana.
- **Mantener el formato:** incluso en equipos pequeños, 15 minutos de sincronización diaria pueden ser valiosos.
- **Lo importante no es el formato sino el principio:** mantener la operativa visible y detectar impedimentos a tiempo.

La daily como revisión de código de IA

Una práctica emergente es usar parte del tiempo de la daily para revisar colectivamente código generado por IA que requiere discusión. No todo el código, pero sí el que plantea dudas o decisiones arquitectónicas.

Esto transforma la daily de un evento de «qué hice ayer» a un evento de «qué revisamos juntos hoy».

5.3. Review y retrospectiva en el nuevo contexto

La revisión y la retrospectiva son los eventos de inspección y adaptación por excelencia. Con la IA, su importancia aumenta, no disminuye.

La review cuando la IA acelera la entrega

La reunión de revisión del sprint sigue siendo el momento de mostrar el incremento y recoger feedback. Pero cuando la IA permite generar más funcionalidades en menos tiempo, emergen nuevas consideraciones.

- **El riesgo de la sobreproducción:** presentar más features de las que los interesados pueden evaluar con atención. Es preferible revisar menos funcionalidades con profundidad que muchas superficialmente.
- **La pregunta que cambia:** de «¿está bien construido?» a «¿estábamos construyendo lo correcto?». Con la velocidad de la IA, la calidad de la decisión de qué construir importa más que la calidad de la construcción.
- **El foco en el valor:** no celebrar cuántas features se entregaron, sino qué impacto tuvieron (o tendrán) esas features. La review puede incluir datos de uso de features anteriores, no solo demostración de features nuevas.
- **Transparencia sobre la IA:** considerar si tiene sentido ser explícitos sobre qué partes del incremento fueron generadas con IA y qué nivel de revisión recibieron.

La retrospectiva: más necesaria que nunca

Paradójicamente, la retrospectiva es el evento que más riesgo corre de ser eliminado por la presión de velocidad de la IA, y sin embargo es el que más se necesita.

¿Por qué es más necesaria?

- **Es el espacio natural para evaluar:** ¿cómo está funcionando la colaboración con la IA?
- **Permite detectar:** ¿estamos acumulando deuda técnica?
- **Plantea:** ¿la velocidad de producción se está traduciendo en valor real?
- **Cuestiona:** ¿nuestros roles siguen teniendo sentido?
- **Decide:** ¿necesitamos adaptar la DoD?

Los equipos que eliminan la retrospectiva para «ir más rápido» pierden el mecanismo que les permite inspeccionar y adaptar su forma de trabajar con IA. Esto es, literalmente, renunciar al empirismo.

Temas específicos para retrospectivas con IA

Las retrospectivas pueden incluir reflexiones específicas sobre el uso de la IA:

- ¿Qué funcionó bien en nuestra colaboración con la IA este sprint?
- ¿Qué no funcionó? ¿Qué código de IA tuvimos que rehacer?
- ¿Estamos revisando el código de IA con suficiente profundidad?
- ¿La calidad de nuestras specs está mejorando?
- ¿Nuestra DoD sigue siendo adecuada?
- ¿Qué competencias necesitamos desarrollar?

Estas preguntas mantienen la retrospectiva relevante para el nuevo contexto.

Frecuencia de la retrospectiva

En flujos muy rápidos, la retrospectiva tradicional de fin de sprint puede no ser suficiente.

Algunas prácticas:

- **Mini-retrospectivas:** 15 minutos al final del día o de la semana para ajustes rápidos.
- **Retrospectivas temáticas:** dedicar una retro al mes específicamente al uso de la IA.
- **Retrospectivas de incidentes:** cuando algo sale mal con código de IA, hacer una retrospectiva específica.

5.4. El equilibrio entre velocidad y reflexión

La tensión fundamental que atraviesa todos los eventos es el equilibrio entre velocidad y reflexión. La IA empuja hacia la velocidad; los eventos empujan hacia la reflexión.

Por qué la reflexión es más difícil con la IA

La IA genera una sensación de urgencia: si podemos producir más rápido, ¿por qué pararnos a reflexionar? Cada hora de retrospectiva es una hora que no estamos generando código.

Esta lógica es tentadora pero peligrosa. La reflexión no es tiempo perdido; es lo que permite que el tiempo de producción sea efectivo.

Lo que la reflexión proporciona

Los eventos de reflexión (especialmente la retrospectiva) proporcionan:

- **Detección temprana de problemas:** identificar que estamos acumulando deuda técnica antes de que sea inmanejable.

- **Aprendizaje colectivo:** compartir qué funciona y qué no en el uso de la IA.
- **Alineación:** asegurar que todos entienden cómo trabajamos y por qué.
- **Mejora continua:** ajustar el proceso basándose en la experiencia.
- **Cohesión de equipo:** los rituales compartidos construyen equipo.

Sin estos beneficios, la velocidad se convierte en velocidad ciega.

El patrón del equipo que fracasa con IA

Un patrón que se repite en equipos que fracasan con IA:

1. Adoptan herramientas de IA.
2. La velocidad de producción aumenta.
3. Eliminan o reducen eventos «para ir más rápido».
4. Pierden la capacidad de detectar problemas.
5. Acumulan deuda técnica sin darse cuenta.
6. La calidad se degrada.
7. El sistema se vuelve frágil.
8. Un incidente grave expone todos los problemas acumulados.
9. El equipo pierde la confianza en la IA.

El punto crítico es el paso 3: eliminar los momentos de reflexión. Una vez perdidos, es difícil detectar los problemas que se acumulan en los pasos siguientes.

El patrón del equipo que prospera con IA

En contraste, los equipos que prosperan:

1. Adoptan herramientas de IA.
2. La velocidad de producción aumenta.
3. Mantienen (o intensifican) los eventos de reflexión.
4. Usan las retrospectivas para ajustar cómo trabajan con IA.
5. Detectan y corrigen problemas temprano.
6. La calidad se mantiene o mejora.
7. El sistema permanece robusto.
8. La confianza en la IA se basa en evidencia, no en fe.

La diferencia está en proteger deliberadamente el tiempo de reflexión.

Cómo proteger la reflexión

Algunas prácticas que ayudan:

- **Tratar los eventos como inamovibles:** no se cancelan por «falta de tiempo».
- **Vincular los eventos a los resultados:** mostrar cómo las decisiones de retrospectivas han mejorado el trabajo.

- **Adaptar el formato, no eliminar:** si la retrospectiva de 2 horas no funciona, probar 1 hora. Pero no cero horas.
- **Incluir métricas:** en la retrospectiva, revisar las métricas de salud del código. Los datos hacen visible lo que la velocidad oculta.

Ejercicios prácticos

Ejercicio 1: evaluación de tu cadencia actual

Reflexiona sobre la cadencia de sprints de tu equipo:

1. Duración actual del sprint: _____
2. ¿Cuánto tiempo pasa típicamente entre que se identifica una necesidad y está en producción?
    a. **Si es mucho menor que la duración del sprint:** la cadencia puede ser demasiado lenta.
    b. **Si es similar:** la cadencia está alineada.
    c. **Si es mayor:** hay otros cuellos de botella además del sprint
3. ¿Los eventos del sprint (planning, review, retro) se sienten útiles o burocráticos?
4. ¿Qué proporción del sprint se dedica a eventos vs. trabajo de construcción?
5. Con la IA, ¿ha cambiado la velocidad a la que generáis incrementos?
6. Si la velocidad ha cambiado, ¿debería cambiar la cadencia?

Opciones a considerar:

- ☐ Mantener la cadencia actual (funciona bien).
- ☐ Acortar los sprints (queremos feedback más frecuente).
- ☐ Añadir micro-iteraciones dentro del sprint (queremos velocidad sin perder el ritmo).
- ☐ Explorar flujo continuo (la cadencia fija ya no tiene sentido).
- ☐ Adoptar cadencia dual (diferentes ritmos para diferentes tipos de trabajo).

¿Qué experimento podrías proponer en la próxima retrospectiva?

Ejercicio 2: rediseño de la planning para IA

Diseña cómo sería una planning adaptada al trabajo con IA:

Planning actual (describe brevemente cómo es)

- Duración:
- Participantes:
- Actividades principales:
- Planning adaptada:
    a. **Tiempo para specs vs. tiempo para historias:**

- ¿Qué proporción del backlog requiere specs detalladas?

b. **Nuevas preguntas a incluir:**
   - ¿Qué tareas delegaremos a la IA?
   - ¿Qué nivel de revisión aplicaremos?
   - ¿Cómo validaremos lo que construyamos?
   - (otras)

c. **Estimación:**
   - ¿Cómo estimaremos el esfuerzo cuando la IA genera parte del código?

d. **Criterios de selección:**
   - ¿Qué criterios usamos para decidir qué entra en el sprint sabiendo que podemos producir más?

e. **Duración propuesta:**
   - ¿Qué cambiaría en la experiencia del equipo?

Ejercicio 3: diseño de retrospectiva sobre IA

Diseña una retrospectiva específica para evaluar cómo está funcionando el trabajo con IA:

1. **Objetivo:** evaluar y mejorar la colaboración del equipo con la IA
2. **Duración:** _____ minutos
3. **Actividades:**
   a. **Check-in (5 min):** "En una palabra, ¿cómo describirías tu experiencia trabajando con IA este sprint?"
   b. **Datos (10 min):** revisar métricas:
      - % de código generado por IA
      - Tiempo de revisión dedicado
      - Bugs encontrados en código de IA
      - Tendencia de duplicación
      - (otros)
4. **Reflexión individual (5 min):**
   a. ¿Qué funcionó bien en mi interacción con la IA?
   b. ¿Qué no funcionó?
   c. ¿Qué aprendí?
5. **Discusión grupal (15 min):** preguntas a explorar:
   a. ¿Nuestra DoD sigue siendo adecuada?
   b. ¿Estamos revisando con suficiente profundidad?
   c. ¿La calidad de nuestras specs mejora?
   d. ¿Qué competencias necesitamos desarrollar?
6. **Acciones (10 min):**
   a. Decidir 1-2 acciones concretas para el próximo sprint.
7. **Check-out (5 min):**

a. "¿Qué me llevo de esta retrospectiva?"
b. ¿Cuándo podrías facilitar esta retrospectiva con tu equipo?

Resumen del capítulo

En este capítulo hemos explorado cómo evolucionan los eventos de scrum:

- La cadencia del sprint entra en tensión con la velocidad de la IA. Las opciones incluyen mantener sprints con micro-iteraciones, sprints más cortos, flujo continuo o cadencia dual. Lo importante es que sea una decisión consciente.
- La planning incorpora nuevas preguntas: qué tareas delegamos a la IA, cómo validamos lo que construimos, qué nivel de especificación necesitamos. La estimación se complica cuando la relación esfuerzo-resultado ya no es estable.
- La daily en equipos pequeños puede evolucionar hacia formatos más ligeros, pero el principio (operativa visible) se mantiene. Puede incluir revisión colectiva de código de IA.
- La review cambia el foco de «¿está bien construido?» a «¿construimos lo correcto?». La retrospectiva se vuelve más necesaria, no menos, como mecanismo para evaluar la colaboración con IA.
- El equilibrio entre velocidad y reflexión es la tensión fundamental. Los equipos que prosperan protegen deliberadamente los momentos de reflexión.

En el próximo capítulo exploraremos los modelos emergentes de trabajo: doble carril, Spec Driven Development, IA como teammate.

Preguntas de autoevaluación

1. ¿Cuáles son las cuatro opciones principales que los equipos están explorando para la cadencia del sprint con IA?
2. ¿Qué nuevas preguntas debe incorporar la planning cuando el equipo trabaja con IA?
3. ¿Por qué la retrospectiva se vuelve 'más necesaria, no menos' con la IA?
4. Describe el 'patrón del equipo que fracasa con IA' y el punto crítico de ese patrón.
5. ¿Qué temas específicos debería incluir una retrospectiva sobre el uso de IA?

CAPÍTULO 6 — Modelos emergentes de trabajo

Más allá de la adaptación de las prácticas clásicas de scrum, la comunidad ágil está desarrollando modelos de trabajo completamente nuevos para integrar la IA. Ninguno de ellos ha alcanzado la madurez de las prácticas consolidadas; todos son propuestas emergentes en evolución. En este capítulo exploraremos los tres modelos más significativos—Doble carril, Spec-Driven Development, e IA como teammate— y cómo combinarlos según el contexto de cada equipo.

6.1. Doble carril: vibe coding y vibe engineering

El modelo de *Dual track* (doble carril) propone separar explícitamente dos flujos de trabajo con objetivos y estándares diferentes. Es la respuesta a una realidad: no todo el trabajo requiere el mismo nivel de rigor, y forzar un único estándar frena unas cosas o compromete otras.

Vibe coding: el carril rápido

Es un flujo ultra rápido orientado a crear prototipos no liberables a producción. Su propósito es validar ideas con usuarios rápidamente, usando herramientas de IA y no-code para generar artefactos funcionales en horas.

Características:

- Velocidad sobre calidad del código.
- Prototipos desechables, no código de producción.
- Experimentación libre con la IA.
- Mínima revisión formal.
- Foco en aprendizaje, no en entrega.

En este carril, la calidad del código es secundaria. Lo que importa es la velocidad de aprendizaje. Es el espacio natural del «vibe coding»: interacción libre y exploratoria con la IA para probar posibilidades.

Vibe engineering: el carril robusto

Es un flujo que asegura mínimos de ingeniería, seguridad y calidad para que el producto sea robusto y mantenible. Aquí se aplican las prácticas de Spec-Driven Development, se refuerza la Definition of Done y se ejecutan los mecanismos de QA.

Características:

- Calidad y mantenibilidad como prioridad.
- Código destinado a producción.
- Specs detalladas antes de generar.

- Revisión exhaustiva del output de IA.
- Foco en robustez, no en velocidad.

Es el espacio donde el código generado por IA se somete al rigor de producción.

La separación necesaria

La separación entre ambos carriles no es nueva —el concepto de «dual-track agile» (discovery + delivery) existe desde hace años—, pero la IA la hace más explícita y necesaria.

¿Por qué? Porque la velocidad del vibe coding puede ser 10x-100x mayor que la del vibe engineering. Sin separación, ocurre una de dos cosas:

- El estándar de producción frena la experimentación: No puedes validar ideas rápidamente si cada prototipo requiere specs, revisiones y DoD completa.
- El estándar de experimentación contamina producción: Si el código de «prueba rápida» acaba en producción sin pasar por el carril robusto, acumulas deuda técnica.

La transición entre carriles

Una idea validada en vibe coding no se traslada directamente a producción. Se reescribe —o se regenera con mejores specs— en vibe engineering. El código del prototipo es descartable; lo que se conserva es el aprendizaje.

Esta distinción es crucial: el prototipo demuestra que la idea funciona; el código de producción implementa la idea correctamente.

Cuándo usar cada carril

Vibe coding es apropiado para:

- Validar hipótesis de producto con usuarios.
- Explorar alternativas técnicas rápidamente.
- Crear demos para stakeholders.
- Probar la viabilidad de una idea antes de invertir.

Vibe engineering es obligatorio para:

- Cualquier código que irá a producción.
- Funcionalidades críticas para el negocio.
- Código que otros desarrolladores mantendrán.
- Sistemas con requisitos de seguridad o compliance.

6.2. Spec-Driven Development: flujo completo

El Spec-Driven Development (SDD) es el modelo más estructurado para trabajar con IA generativa. Ya lo introdujimos en el capítulo 4; aquí profundizamos en su implementación práctica.

El flujo completo

El SDD sigue cuatro fases que se ejecutan de manera iterativa:

FASE 1: SPECIFY

El humano describe a alto nivel qué quiere construir y por qué. Esta descripción inicial es más rica que una historia de usuario pero no es aún una spec completa.

- **Input:** descripción de alto nivel del problema y la solución deseada.
- **Proceso:** la IA genera una especificación detallada basándose en la descripción. El humano revisa, ajusta y valida.
- **Output:** especificación detallada con contexto, requisitos, criterios de aceptación y ejemplos.

FASE 2: PLAN

La IA propone un plan de implementación basado en la spec.

- **Input:** la especificación validada.
- **Proceso:** la IA analiza la spec y propone cómo implementarla: qué componentes crear o modificar, en qué orden, con qué dependencias.
- **Output:** plan de implementación estructurado.

FASE 3: TASKS

La spec y el plan se descomponen en tareas pequeñas.

- **Input:** especificación y plan de implementación.
- **Proceso:** se identifican tareas individuales lo suficientemente pequeñas para ser implementadas y revisadas de forma aislada.
- **Output:** lista de tareas concretas y acotadas.

FASE 4: IMPLEMENT

La IA ejecuta las tareas una a una.

- **Input:** una tarea específica con su contexto.
- **Proceso:** la IA genera el código. El desarrollador revisa el cambio focalizado.
- **Output:** código revisado y aprobado para esa tarea.

La iteración

El flujo no es lineal. En cualquier fase puede descubrirse que la spec necesita ajustes, que el plan no funciona, o que una tarea revela un problema en la concepción original. El modelo acepta y espera esta iteración.

Niveles de adopción

No todos los equipos necesitan adoptar SDD al máximo nivel:

- **Nivel básico:** escribir specs antes de pedir código a la IA, aunque las specs no se ejecuten automáticamente.
- **Nivel intermedio:** incluir en las specs criterios de aceptación que se convierten en tests automatizados.
- **Nivel avanzado:** tratar la spec como fuente de verdad, regenerando el código desde la spec cuando sea necesario.

Herramientas

Herramientas como GitHub Spec Kit, Kiro y Tessl están formalizando este flujo. Proporcionan:

- Plantillas para specs.
- Integración con agentes de código.
- Seguimiento de la relación spec-código.
- Validación automática de que el código cumple la spec.

Limitaciones actuales

Es importante ser realista: el SDD es una práctica emergente. Los agentes de IA aún no siguen todas las instrucciones de las specs de manera fiable. La calidad del output depende enormemente de la calidad de la spec, y escribir buenas specs es una competencia que se desarrolla con práctica.

6.3. IA como teammate: el framework de cuatro pasos

El framework «IA como teammate» de Scrum.org propone un enfoque más conservador y gradual para integrar la IA en equipos scrum. Es compatible con las prácticas habituales y puede adoptarse de manera incremental.

El concepto central

En lugar de tratar la IA como una herramienta más, se la trata como un compañero de equipo al que se hace «onboarding»: se le proporciona contexto, se le establecen límites, y se le integra en los flujos de trabajo.

La analogía no es perfecta (la IA no tiene motivación ni responsabilidad), pero es útil para estructurar la adopción.

Los cuatro pasos

PASO 1: MODEL MANAGEMENT

Selección y gestión de los modelos de IA adecuados para cada tarea del equipo.

Preguntas clave:

- ¿Qué modelos de IA usaremos? (GPT-4, Claude, Codex, modelos especializados...)
- ¿Para qué tareas usaremos cada uno?
- ¿Cómo gestionamos las versiones y actualizaciones de los modelos?
- ¿Qué costes implica y cómo los controlamos?

Salida: catálogo de modelos disponibles y sus usos recomendados.

PASO 2: CONTEXT MANAGEMENT

Proporcionar a la IA el contexto necesario para que sus outputs sean relevantes.

Preguntas clave:

- ¿Qué necesita saber la IA sobre nuestra arquitectura?
- ¿Cómo le proporcionamos los estándares de código del equipo?
- ¿Cómo incluimos el historial de decisiones técnicas?
- ¿Cómo actualizamos el contexto cuando cambia?

Salida: «base de conocimiento» que se incluye en las interacciones con la IA.

PASO 3: PROMPT ENGINEERING

Desarrollar la competencia de comunicación efectiva con la IA como disciplina del equipo.

Preguntas clave:

- ¿Cómo escribimos prompts efectivos?
- ¿Tenemos plantillas o patrones compartidos?
- ¿Cómo documentamos qué prompts funcionan bien?
- ¿Cómo formamos a nuevos miembros en esta competencia?

Salida: guía de prompts del equipo, ejemplos documentados, formación.

PASO 4: GOVERNANCE MANAGEMENT

Establecer políticas de uso, límites de gasto, auditoría de outputs y mecanismos de rendición de cuentas.

Preguntas clave:

- ¿Qué límites ponemos al uso de IA? (costes, tipos de tareas, datos sensibles...)
- ¿Cómo auditamos el código generado?
- ¿Quién es responsable cuando algo falla?
- ¿Cómo aseguramos el cumplimiento de políticas de la organización?

Salida: políticas de gobernanza documentadas y comunicadas.

Ventajas

Es un enfoque conservador que:

- Se integra con las prácticas existentes de scrum.
- Puede adoptarse paso a paso.
- No requiere cambios radicales en la forma de trabajar.
- Proporciona estructura sin ser prescriptivo.

Es especialmente adecuado como punto de partida para equipos que aún no han formalizado su relación con la IA.

Limitaciones

Este framework no aborda directamente:

- Cómo cambian los roles con la IA.
- Cómo evolucionan los artefactos.
- Modelos de trabajo alternativos como SDD o doble carril.

Es un complemento a estos otros modelos, no un sustituto.

6.4. Modelos híbridos: combinar según contexto

En la práctica, la mayoría de los equipos no adoptarán un único modelo de forma pura, sino que combinarán elementos según su contexto. La clave es entender qué modelo es más adecuado para cada situación.

Combinaciones comunes

Doble carril + SDD

- **Vibe coding:** prototipado libre sin specs formales
- **Vibe engineering:** SDD completo para código de producción Esta combinación aprovecha la velocidad para experimentar y el rigor para producir.

SDD + IA como teammate

- SDD para el flujo de trabajo técnico
- IA como teammate para la gobernanza y el onboarding del equipo Esta combinación estructura tanto el «cómo trabajamos» como el «cómo nos organizamos».

Scrum evolucionado + adopción gradual

- Mantener la estructura de scrum.
- Adoptar elementos de los otros modelos incrementalmente.
- Usar retrospectivas para evaluar qué funciona.
- Apropiado para equipos que no quieren cambios radicales.

Guía de selección por contexto

Producto nuevo, validación temprana	Doble carril con énfasis en vibe coding	Priorizar velocidad de aprendizaje
Producto maduro, base de código extensa	SDD + QA reforzado	Proteger la integridad del sistema
Industria regulada (fintech, salud)	IA como teammate con gobernanza fuerte	Maximizar control y trazabilidad
Startup, equipo muy pequeño (2-3)	Flujo continuo + SDD ligero	Minimizar overhead
Equipo grande con sistemas legados	Scrum evolucionado + adopción incremental	Gestionar transición sin disrupciones

La experimentación disciplinada

La clave no es elegir el modelo «correcto» de antemano, sino experimentar de manera disciplinada:

1. **Hipótesis:** «Creemos que adoptar X mejorará Y».
2. **Experimento:** probar X durante un tiempo definido.

3. **Medición:** recoger datos sobre Y.
4. **Evaluación:** ¿La hipótesis se confirmó?
5. **Decisión:** continuar, ampliar, ajustar o abandonar.

En última instancia, esto es exactamente lo que la comunidad ágil lleva haciendo desde los años 80: evolucionar sus prácticas en respuesta al contexto.

Anti-patrones a evitar

- **Adopción por moda:** adoptar un modelo porque «todo el mundo lo hace» sin evaluar si encaja con el contexto.
- **Todo o nada:** intentar adoptar un modelo completo de golpe en lugar de incrementalmente.
- **Rigidez**: aferrarse a un modelo que no está funcionando en lugar de experimentar con otros.
- **Falta de evaluación:** adoptar prácticas sin definir cómo sabremos si funcionan.

El rol de la retrospectiva

La retrospectiva sigue siendo el mecanismo natural para evaluar qué modelo o combinación está funcionando. Preguntas útiles:

- ¿El modelo actual nos está ayudando o estorbando?
- ¿Qué elementos del modelo funcionan bien?
- ¿Qué elementos deberíamos ajustar o eliminar?
- ¿Hay elementos de otros modelos que deberíamos probar?

Ejercicios prácticos

Ejercicio 1: diseño de doble carril para tu equipo

Diseña cómo funcionaría el modelo de doble carril en tu contexto.

VIBE CODING (carril rápido)

1. ¿Qué tipo de trabajo iría a este carril? (prototipos, demos, experimentos, validaciones...)
2. ¿Qué estándares mínimos aplicarían? (¿alguna revisión? ¿documentación? ¿tests?)
3. ¿Quién puede trabajar en este carril? (¿solo desarrolladores? ¿También PO/diseñadores?)
4. ¿Cuánto tiempo máximo puede estar algo en este carril antes de decidir si pasa a producción o se descarta?

VIBE ENGINEERING (carril robusto)

1. ¿Qué estándares aplicarían? (DoD completa, specs, revisiones, tests...)
2. ¿Cómo sería el proceso de transición desde vibe coding? (¿reescribir? ¿regenerar con mejores specs? ¿refactorizar?)
3. ¿Qué porcentaje del tiempo del equipo estimáis que iría a cada carril?

GOBERNANZA

1. ¿Cómo sabéis en qué carril está cada pieza de trabajo?
2. ¿Cómo evitáis que código de vibe coding acabe en producción sin pasar por vibe engineering?
3. ¿Qué obstáculos anticipáis para implementar este modelo?

Ejercicio 2: implementación gradual de IA como teammate

Planifica cómo adoptarías el framework de IA como teammate en tu equipo.

PASO 1: MODEL MANAGEMENT

- ¿Qué modelos de IA usa actualmente tu equipo?
- ¿Hay algún catálogo o documentación de cuándo usar cada uno?
- ¿Qué falta por definir?
- Acción concreta:

PASO 2: CONTEXT MANAGEMENT

- ¿Proporcionáis contexto a la IA (arquitectura, estándares, historial)?
- ¿Cómo lo hacéis actualmente?
- ¿Qué mejoraría la calidad del output?
- Acción concreta:

PASO 3: PROMPT ENGINEERING

- ¿Tenéis patrones o plantillas compartidas para prompts?
- ¿Documentáis qué prompts funcionan bien?
- ¿Cómo formáis a nuevos miembros?
- Acción concreta:

PASO 4: GOVERNANCE MANAGEMENT

- ¿Hay límites definidos para el uso de IA?
- ¿Quién es responsable del código generado?
- ¿Cómo auditáis el output?
- Acción concreta:

PLAN DE ADOPCIÓN

- ¿Por qué paso empezaríais?
- ¿En qué plazo esperáis tener los cuatro pasos implementados?

Ejercicio 3: selección de modelo para tu contexto

Analiza tu contexto y decide qué modelo o combinación es más apropiado.

ANÁLISIS DEL CONTEXTO

1. **Madurez del producto:**
   - ☐ Nuevo, en fase de validación.
   - ☐ En crecimiento activo.
   - ☐ Maduro, énfasis en mantenimiento.
   - ☐ Legacy, necesita modernización.
2. **Tamaño del equipo:**
   - ☐ Muy pequeño (2-3).
   - ☐ Pequeño (4-6).
   - ☐ Mediano (7-10).
   - ☐ Grande (>10).
3. **Restricciones de la industria:**
   - ☐ Sin restricciones especiales.
   - ☐ Algunas restricciones (calidad, seguridad...).
   - ☐ Industria regulada (fintech, salud, defensa...).
4. **Madurez ágil del equipo:**
   - ☐ Principiante.
   - ☐ Intermedio.
   - ☐ Avanzado.
5. **Adopción actual de IA:**
   - ☐ Ninguna o muy baja.
   - ☐ Uso individual no estandarizado.
   - ☐ Uso estandarizado parcial.
   - ☐ Uso maduro y gobernado.

SELECCIÓN DE MODELO

Basándote en el análisis, ¿qué modelo o combinación parece más apropiado?

- ☐ Doble carril
- ☐ SDD completo
- ☐ IA como teammate
- ☐ Scrum evolucionado con adopción gradual
- ☐ Combinación: _____

JUSTIFICACIÓN

¿Por qué este modelo para tu contexto?

PRIMER EXPERIMENTO

¿Qué experimento concreto harías para probar este modelo?

Resumen del capítulo

En este capítulo hemos explorado los modelos emergentes de trabajo con IA:

- El doble carril separa vibe coding (prototipado rápido, código desechable) de vibe engineering (código de producción con rigor). La velocidad del primero puede ser 10x-100x mayor que el segundo.
- El Spec-Driven Development utiliza especificaciones como fuente de verdad. Sus cuatro fases (Specify, Plan, Tasks, Implement) proporcionan un flujo estructurado. Puede adoptarse a diferentes niveles de madurez.
- El framework IA como teammate de Scrum.org propone cuatro pasos: Model Management, Context Management, Prompt Engineering, Governance Management. Es un enfoque conservador y gradual.
- Los modelos híbridos combinan elementos según el contexto. La clave es la experimentación disciplinada: hipótesis, experimento, medición, evaluación, decisión.

En el próximo capítulo exploraremos los mecanismos de calidad y gobernanza específicos para código generado por IA.

Preguntas de autoevaluación

1. ¿Cuál es la diferencia fundamental entre vibe coding y vibe engineering?
2. ¿Cuáles son las cuatro fases del Spec-Driven Development y qué produce cada una?
3. ¿Cuáles son los cuatro pasos del framework IA como teammate de Scrum.org?
4. ¿Por qué la transición de vibe coding a vibe engineering implica reescribir, no trasladar el código?
5. ¿Qué anti-patrones hay que evitar al seleccionar un modelo de trabajo con IA?

CAPÍTULO 7 — Calidad y gobernanza

La calidad del software es posiblemente el área donde la IA genera más tensión entre sus promesas y su realidad. La velocidad de generación de código puede erosionar la calidad si no hay mecanismos deliberados para contrarrestarlo. En este capítulo profundizamos en los datos sobre degradación del código, los modelos emergentes de QA, la gobernanza del uso de IA, y la transparencia necesaria sobre qué genera la IA y qué revisan los humanos.

7.1. La degradación silenciosa del código

Los datos de la industria muestran patrones preocupantes en proyectos que adoptan IA sin controles deliberados de calidad. Es lo que podemos llamar «degradación silenciosa»: el código funciona, pero se vuelve progresivamente más frágil, duplicado y costoso de mantener.

Los datos

Análisis de grandes bases de código revelan tendencias que merecen atención:

- **Aumento de duplicación:** la duplicación de código en proyectos asistidos por IA aumentó significativamente (del ~8% al ~12% de las líneas cambiadas). La IA tiende a generar código similar para problemas similares, en lugar de reutilizar soluciones existentes.
- **Caída del refactoring:** la actividad de refactoring cayó drásticamente. Los equipos producen código nuevo en lugar de mejorar el existente. Es más fácil pedir a la IA que genere algo nuevo que pedirle que mejore lo que hay.
- **Código generado supera al refactorizado:** en 2024, por primera vez, el código copiado o generado por IA superó al código refactorizado. Este es un cambio estructural en cómo evolucionan las bases de código.
- **Caída de la confianza:** la confianza de los desarrolladores en la precisión del código generado por IA cayó del 69% en 2024 al 54% en 2025. A medida que el uso se generalizó, los equipos experimentaron más los límites de la tecnología.

Por qué ocurre

Estos patrones no son inevitables, pero sí son los resultados por defecto si no hay intervención deliberada:

- **La IA no conoce tu base de código (sin context management):** sin contexto explícito, la IA genera soluciones genéricas que pueden duplicar lo que ya existe.

- **El refactoring no es «sexy»:** pedir a la IA «mejora este código» es menos satisfactorio que pedir «crea esta nueva feature». El sesgo hacia lo nuevo es humano, no de la IA.
- **La revisión es el cuello de botella:** revisar código generado es cognitivamente exigente. Cuando hay presión de tiempo, la revisión se superficializa.
- **La velocidad oculta los problemas:** el código funciona hoy; los problemas de mantenibilidad aparecen mañana. La gratificación es inmediata; el coste es diferido.

La frustración más común

La frustración más reportada por desarrolladores es obtener soluciones que son «casi correctas pero no del todo»: el output parece plausible pero contiene errores sutiles que generan trabajo adicional.

Esto es especialmente problemático porque el código «casi correcto» puede pasar revisiones superficiales y llegar a producción, donde los errores se manifiestan de formas inesperadas.

Implicaciones

Sin controles deliberados, la IA acelera la acumulación de deuda técnica. El equipo va más rápido, pero el código se vuelve frágil, duplicado y costoso de mantener.

La velocidad de hoy se paga con mantenibilidad de mañana. Y en software, el mañana llega rápido.

7.2. QA de nueva generación

El aseguramiento de la calidad cambia cuando la IA genera código. Están surgiendo nuevos enfoques que complementan las prácticas tradicionales.

IA supervisando IA

Un modelo emergente es utilizar una IA como revisor automático de lo que otra IA genera. Esto no sustituye la revisión humana, pero la hace más eficiente:

- **Análisis automático pre-revisión:** antes de que un humano revise, una IA analiza el código generado buscando:
    - Patrones problemáticos conocidos.
    - Duplicación con código existente.
    - Vulnerabilidades de seguridad comunes.
    - Desviaciones de los estándares del equipo.

- **Semáforo de confianza:** la IA proporciona un indicador de confianza: verde (parece bien), amarillo (requiere atención), rojo (problemas detectados). El revisor humano puede priorizar su atención.
- **Sugerencias de mejora:** la IA revisora puede sugerir mejoras que la IA generadora no consideró.
- **Limitaciones:** la IA revisora tiene las mismas limitaciones que la generadora. No puede detectar problemas que requieran comprensión profunda del contexto del negocio o de la arquitectura. Es un filtro, no un sustituto.

Specification By Example (SBE)

La SbE proporciona expectativas concretas y testeables que sirven como «verdad de referencia» para el código generado:

- **El principio:** la IA no puede «corregir sus propios deberes». Necesita una referencia externa que no haya generado ella misma.
- **La práctica:** crear ejemplos concretos de comportamiento esperado ANTES de generar el código. Estos ejemplos se convierten en tests que el código debe pasar.
- **El flujo:**
    1. Humanos definen ejemplos de comportamiento esperado.
    2. Estos ejemplos se formalizan como tests automatizados.
    3. La IA genera código.
    4. El código se ejecuta contra los tests.
    5. Si falla, se itera; si pasa, se revisa manualmente
- **Ventaja:** los tests son la «fuente de verdad» independiente. El código de IA no puede pasar tests que no cumpla realmente.

Tests generados por IA

Una práctica complementaria es pedir a la IA que genere tests, pero con precauciones:

- **Riesgo:** si la misma IA genera el código y los tests, puede haber «colusión» (los tests verifican lo que el código hace, no lo que debería hacer).
- **Mitigación:** usar diferentes contextos o incluso diferentes modelos para generar código y tests. O generar tests antes del código (TDD con IA).

Auditoría continua

Más allá de la revisión por PR, algunos equipos implementan auditoría continua de la base de código:

- **Métricas monitorizadas:**
    - Tendencia de duplicación.
    - Complejidad ciclomática media.
    - Cobertura de tests.

    - Ratio código nuevo vs. refactorizado.
    - Deuda técnica identificada.
- **Alertas:** cuando una métrica cruza un umbral, se dispara una alerta. Esto permite detectar degradación antes de que sea grave.
- **Dashboard de salud:** visibilidad continua del estado de la base de código, no solo en momentos puntuales.

7.3. Gobernanza del uso de IA

La gobernanza establece las reglas, límites y responsabilidades del uso de IA en el equipo. Sin gobernanza explícita, cada persona hace lo que considera apropiado, con resultados inconsistentes.

Políticas de uso

Las políticas definen qué está permitido, qué está prohibido y qué requiere supervisión:

¿Para qué tareas se puede usar IA?

- Generación de código: ¿siempre? ¿con restricciones? ¿para qué tipos de código?
- Generación de tests: ¿permitida? ¿con qué controles?
- Documentación: ¿permitida? ¿quién la revisa?Análisis de datos: ¿qué datos pueden procesarse con IA externa?
- ¿Qué datos pueden enviarse a la IA?
- Código propietario: ¿puede enviarse a APIs externas?
- Datos de clientes: ¿nunca? ¿anonimizados?
- Información confidencial: ¿qué se considera confidencial?

¿Qué modelos están aprobados?

- ¿Hay modelos corporativos preferidos?
- ¿Se pueden usar herramientas gratuitas?
- ¿Quién aprueba nuevas herramientas?

Límites de gasto

El uso de IA tiene costes que pueden escalar rápidamente:

- Presupuesto por equipo/mes.
- Alertas cuando el consumo supera umbrales.
- Revisión de uso en retrospectivas.
- Responsable de aprobar excepciones.

Responsabilidad y rendición de cuentas

¿Quién es responsable cuando algo falla?

- **Código en producción:** el código generado por IA que llega a producción es responsabilidad del equipo, no de la IA. Alguien lo revisó (o debería haberlo revisado) y lo aprobó.
- **Errores de la IA:** si la IA genera código con vulnerabilidades, ¿quién responde? El revisor que lo aprobó, el equipo que estableció (o no) los controles.
- **Incidentes:** cuando hay un incidente relacionado con código de IA, ¿cómo se investiga? ¿Se incluye el análisis de por qué la revisión no lo detectó?

Auditoría

¿Cómo se verifica el cumplimiento de las políticas?

- **Trazabilidad:** ¿Se puede saber qué código fue generado por IA?
- **Revisión periódica:** ¿Se revisa regularmente el cumplimiento de las políticas?
- **Métricas de gobernanza:** ¿Se miden indicadores como % de código de IA que pasa por revisión, tiempo medio de revisión, incidentes relacionados con código de IA?

Evolución de las políticas

Las políticas no deben ser estáticas:

- Revisión regular (trimestral, por ejemplo).
- Incorporación de aprendizajes de incidentes.
- Adaptación a nuevas herramientas y capacidades.
- Input del equipo en retrospectivas.

7.4. Transparencia: qué genera la IA y qué revisan los humanos

La transparencia sobre el uso de IA es un valor emergente que algunos equipos están adoptando deliberadamente. Implica ser explícitos sobre qué partes del trabajo fueron generadas con IA y qué nivel de revisión recibieron.

Por qué importa la transparencia

Para el equipo

- Permite evaluar la calidad del proceso de revisión.
- Facilita identificar patrones (¿los bugs vienen más del código de IA o del código humano?).
- Apoya el aprendizaje sobre qué funciona y qué no.

Para los stakeholders

- Pueden entender cómo se construye el producto.
- Pueden calibrar sus expectativas.
- Pueden participar en decisiones sobre niveles de riesgo aceptables.

Para la organización

- Visibilidad del grado de adopción de IA.
- Datos para decisiones sobre inversión en herramientas.
- Base para políticas de gobernanza.

Prácticas de transparencia

Etiquetado de código

- Marcar en commits qué código fue generado por IA.
- Indicar quién lo revisó y qué tipo de revisión se aplicó.
- Mantener esta información en el historial.

Métricas visibles

- Dashboard con % de código generado por IA por sprint/proyecto.

Tendencias de calidad segmentadas por origen del código

- Tiempo de revisión por tipo de código.
- Comunicación en reviews:
    - En la revisión del sprint, mencionar qué funcionalidades usaron IA significativamente.
    - Compartir aprendizajes sobre qué funcionó bien y qué no.

Documentación de decisiones

- Registrar cuándo se decide usar o no usar IA para una tarea.
- Documentar los criterios de esa decisión.

Límites de la transparencia

La transparencia total puede tener efectos no deseados:

- **Estigma:** si el código de IA se percibe como «de segunda», los equipos pueden evitar etiquetarlo aunque lo usen.
- **Falsa precisión:** la línea entre «código de IA» y «código humano» es borrosa. Un humano que usa autocompletado de IA, ¿está generando código de IA o código humano?

- **Overhead:** mantener metadatos detallados tiene un coste.

La recomendación es encontrar un nivel de transparencia que aporte valor sin generar burocracia ni efectos perversos.

Hacia una cultura de la transparencia

Más allá de las prácticas específicas, lo importante es cultivar una cultura donde:

- Usar IA no es vergonzoso ni problemático.
- No usar IA cuando sería útil tampoco es heroico.
- Las decisiones sobre el uso de IA se toman conscientemente.
- Los aprendizajes se comparten abiertamente.
- Los errores (propios o de la IA) son oportunidades de mejora.

Esta cultura se construye con el ejemplo del liderazgo y se refuerza en las retrospectivas.

Ejercicios prácticos

Ejercicio 1: análisis de salud del código

Evalúa el estado de salud de tu base de código en relación con el uso de IA:

MÉTRICAS A RECOPILAR (si tienes acceso a ellas)

1. Duplicación de código:
   - Nivel actual: _____%
   - Tendencia últimos 6 meses:
     - ☐ Aumentando.
     - ☐ Estable.
     - ☐ Disminuyendo.
2. Cobertura de tests:
   - Nivel actual: _____%
   - Tendencia:
     - ☐ Aumentando.
     - ☐ Estable.
     - ☐ Disminuyendo.
3. Complejidad ciclomática media:
   - Nivel actual: _____
   - Tendencia:
     - ☐ Aumentando.
     - ☐ Estable.
     - ☐ Disminuyendo.
4. Bugs en producción (últimos 3 meses):

- Total: _____
- Relacionados con código de IA (si lo sabéis): _____

EVALUACIÓN CUALITATIVA (si no tienes métricas)

➔ ¿El código es más difícil de entender qué hace un año?
- ☐ Sí.
- ☐ No.
- ☐ No sé.

➔ ¿Hay más código duplicado del que os gustaría?
- ☐ Sí.
- ☐ No.
- ☐ No sé.

➔ ¿Los nuevos miembros tardan más en entender la base de código?
- ☐ Sí.
- ☐ No.
- ☐ No sé.

➔ ¿Refactorizáis regularmente o solo añadís código nuevo?
- ☐ Refactorizamos.
- ☐ Solo añadimos.

CONCLUSIONES

- ¿Hay señales de degradación silenciosa?
- ¿Qué métricas deberíais empezar a monitorizar?
- ¿Qué acción podríais tomar en el próximo sprint?

Ejercicio 2: diseño de políticas de gobernanza

Diseña las políticas de gobernanza del uso de IA para tu equipo:

POLÍTICAS DE USO

1. ¿Para qué tareas está permitido usar IA?
   - ☐ Generación de código: _____
   - ☐ Generación de tests: _____
   - ☐ Documentación: _____
   - ☐ Otras: _____
2. ¿Qué datos pueden enviarse a IA externa?
   - ☐ Código propietario: _____
   - ☐ Datos de clientes: _____
   - ☐ Información confidencial: _____
3. ¿Qué modelos/herramientas están aprobados? _____

REVISIÓN Y RESPONSABILIDAD

4. ¿Qué nivel de revisión requiere el código generado por IA? _____
5. ¿Quién es responsable del código de IA en producción? _____
6. ¿Cómo se investigan incidentes relacionados con código de IA? _____

LÍMITES Y CONTROL

7. ¿Hay límites de gasto? ¿Cuáles? _____
8. ¿Cómo se verifica el cumplimiento de estas políticas? _____
9. ¿Con qué frecuencia se revisan estas políticas? _____
10. ¿Cómo comunicarías estas políticas al equipo?

Ejercicio 3: plan de transparencia

Define qué nivel de transparencia sobre el uso de IA quieres para tu equipo:

PREGUNTAS CLAVE

1. ¿Queréis saber qué código fue generado por IA?
   - ☐ No es necesario.
   - ☐ Sería útil pero no prioritario.
   - ☐ Es importante saberlo.
   - ☐ Es crítico saberlo.
2. ¿Cómo etiquetaríais el código de IA?
   - ☐ En los commits.
   - ☐ En comentarios del código.
   - ☐ En un sistema separado.
   - ☐ No lo etiquetaríamos.
3. ¿Qué métricas de uso de IA os gustaría tener visibles?
   - ☐ % de código generado por IA.
   - ☐ Tiempo de revisión de código de IA.
   - ☐ Bugs por origen del código.
   - ☐ Otras: _____
4. ¿Comunicaríais en las reviews del sprint qué usó IA?
   - ☐ Sí, siempre.
   - ☐ Solo si es significativo.
   - ☐ No, no es relevante.
5. ¿Qué beneficios esperáis de esta transparencia? _____
6. ¿Qué riesgos veis (estigma, overhead, etc.)? _____

PLAN DE IMPLEMENTACIÓN:

- ¿Qué prácticas de transparencia adoptaríais primero?

- ¿Cómo evaluaríais si aportan valor?

Resumen del capítulo

En este capítulo hemos explorado la calidad y gobernanza del código generado por IA:

- **La degradación silenciosa es un riesgo real:** aumento de duplicación, caída del refactoring, erosión de la confianza. Sin controles deliberados, la velocidad erosiona la calidad.
- El QA de nueva generación incluye IA supervisando IA (análisis automático prerevisión), Specification by Example (tests como verdad de referencia independiente), y auditoría continua de métricas de salud.
- La gobernanza establece políticas de uso, límites de gasto, responsabilidades y mecanismos de auditoría. Las políticas deben revisarse y evolucionar regularmente.
- La transparencia sobre qué genera la IA y qué revisan los humanos permite evaluar el proceso, identificar patrones y tomar decisiones informadas. Debe equilibrarse con el overhead que genera.

En el último capítulo exploraremos cómo liderar la evolución en tu equipo u organización.

Preguntas de autoevaluación

1. ¿Cuáles son los principales indicadores de 'degradación silenciosa' del código según los datos de la industria?
2. ¿Qué es 'IA supervisando IA' y cuáles son sus limitaciones?
3. ¿Por qué la Specification by Example es especialmente útil para código generado por IA?
4. ¿Qué elementos debe incluir una política de gobernanza del uso de IA?
5. ¿Cuáles son los beneficios y riesgos de la transparencia sobre el uso de IA en el equipo?

CAPÍTULO 8 — Liderando la evolución

Hemos recorrido el mapa del nuevo terreno: el cambio de paradigma, los principios como brújula, la transformación de roles, artefactos y eventos, los modelos emergentes y los mecanismos de calidad. Ahora la pregunta es: ¿cómo lideramos esta evolución en nuestro equipo u organización?

En este capítulo final ofrecemos herramientas prácticas para facilitar la conversación, experimentar disciplinadamente, identificar el valor diferencial humano y construir tu propio plan de desarrollo profesional.

8.1. Canvas de autoevaluación para el equipo

En lugar de un plan de implementación prescriptivo, ofrecemos un canvas de autoevaluación que cada equipo puede utilizar como herramienta de reflexión. Se recomienda usarlo en una sesión facilitada donde todos los miembros puedan contribuir.

Las cinco dimensiones

El canvas evalúa cinco dimensiones clave:

1. Cadencia

- ¿Dónde estamos? ¿Nuestros sprints se sienten útiles o lentos? ¿Entregamos antes de que termine el sprint?
- ¿Qué fricción sentimos? ¿La IA nos permite generar incrementos más rápido de lo que nuestro ciclo permite integrar?
- ¿Qué podemos experimentar? Probar sprints de 1 semana, o micro-iteraciones dentro del sprint actual.

2. Composición del equipo

- ¿Dónde estamos? ¿Cuántas personas usan IA activamente? ¿Qué tareas ha absorbido la IA?
- ¿Qué fricción sentimos? ¿Hay roles cuya aportación se ha diluido? ¿Los eventos son demasiado pesados para nuestro tamaño?
- ¿Qué podemos experimentar? Simplificar un evento específico. Definir explícitamente qué hace la IA y qué los humanos.

3. Lenguaje del trabajo

- ¿Dónde estamos? ¿Usamos historias de usuario, specs, prompts libres? ¿La IA entiende bien lo que le pedimos?

- ¿Qué fricción sentimos? ¿El output de la IA se desvía de nuestra intención? ¿Pasamos mucho tiempo corrigiendo?
- ¿Qué podemos experimentar? Pilotar SDD en una funcionalidad. Crear una plantilla de spec para el equipo.

4. Roles

- ¿Dónde estamos? ¿Cómo ha cambiado el día a día de cada rol desde que usamos IA?
- ¿Qué fricción sentimos? ¿Algún rol siente que su trabajo es redundante? ¿Falta alguien que valide estratégicamente?
- ¿Qué podemos experimentar? Redefinir la aportación de valor de cada rol. Invertir tiempo del facilitador en gobernanza de IA.

5. Calidad y QA

- ¿Dónde estamos? ¿Tenemos métricas de calidad del código generado por IA? ¿Nuestra DoD contempla la IA?
- ¿Qué fricción sentimos? ¿Estamos acumulando deuda técnica más rápido? ¿Confiamos en el código generado?
- ¿Qué podemos experimentar? Adaptar la DoD. Implementar revisión automatizada. Medir duplicación.

Cómo usar el canvas

1. **Evaluar:** cada miembro responde individualmente a «¿dónde estamos?»
2. **Compartir:** se ponen en común las respuestas, identificando patrones y divergencias.
3. **Priorizar:** se seleccionan las 1-2 dimensiones donde la fricción es más evidente.
4. **Experimentar:** se diseña un experimento concreto para el próximo ciclo.
5. **Inspeccionar y adaptar:** se revisan los resultados en la retrospectiva y se decide si continuar, ampliar o pivotar.

8.2. Cómo facilitar la conversación de adaptación

La conversación sobre cómo adaptar las prácticas a la IA puede ser incómoda. Toca temas sensibles: roles que cambian, competencias que se vuelven menos relevantes, formas de trabajar que se cuestionan.

Principios para facilitar

- **Curiosidad sobre juicio:** abordar la conversación con genuina curiosidad sobre qué está funcionando y qué no, en lugar de con juicios predeterminados sobre qué «debería» ser.

- **Datos sobre opiniones:** cuando sea posible, partir de datos concretos (métricas, hechos observables) en lugar de percepciones. «El 40% de nuestro código es generado por IA» es más útil que «usamos mucho IA».
- **Incluir todas las voces:** asegurar que todos los roles tienen espacio para expresarse, especialmente aquellos que pueden sentirse amenazados por el cambio.
- **Enfoque en el equipo:** la conversación es sobre cómo el equipo puede funcionar mejor, no sobre individuos específicos.
- **Experimentación, no decisiones finales:** proponer experimentos reversibles reduce la ansiedad sobre tomar decisiones «incorrectas».

Preguntas poderosas

Algunas preguntas que pueden abrir conversaciones productivas:

Sobre el estado actual

- «¿Cómo describiríais nuestro uso actual de IA en una frase?».
- «¿Qué ha cambiado en vuestro día a día desde que usamos IA?».
- «¿Dónde sentís que la IA nos ayuda más? ¿Dónde nos estorba?».

Sobre las tensiones

- «¿Qué prácticas actuales se sienten desajustadas con la velocidad que la IA permite?».
- «¿Hay momentos donde os sentís presionados a ir más rápido de lo que consideráis prudente?».
- «¿Qué os preocupa del uso de IA que no hemos hablado abiertamente?».

Sobre el futuro

- «Si pudiéramos diseñar nuestra forma de trabajar desde cero con IA, ¿qué mantendríamos y qué cambiaríamos?».
- «¿Qué competencias sentís que necesitáis desarrollar?».
- «¿Qué experimento os gustaría probar en el próximo sprint?».

Gestionar la resistencia

Es normal que haya resistencia al cambio. Algunas formas de abordarla:

- **Validar las preocupaciones:** «Entiendo que preocupa que X cambie. ¿Qué tendríamos que ver para que te sintieras más cómodo?»
- **Hacer el riesgo explícito:** «¿Qué es lo peor que podría pasar si probamos esto durante un sprint? ¿Es reversible?»

- **Buscar pequeños wins:** empezar por cambios pequeños que demuestren valor antes de proponer cambios mayores.
- **Dar tiempo:** no todos procesan el cambio al mismo ritmo. Respetar que algunos necesitan más tiempo.

8.3. El valor diferencial humano

Una pregunta subyacente a toda la evolución con IA es: ¿cuál es el valor diferencial de los humanos? Si la IA puede generar código, escribir tests, analizar datos y proponer soluciones, ¿qué queda para nosotros?

La respuesta es: precisamente lo que la IA no puede hacer. Y eso es más valioso, no menos.

Lo que la IA no puede hacer

- Juicio contextual: **la IA puede procesar información, pero el juicio sobre qué** información importa, cómo interpretarla en el contexto específico y qué decisión tomar requiere experiencia humana.
- **Empatía genuina:** la IA puede simular empatía, pero no puede sentir genuinamente lo que siente otra persona. La conexión humana real —con compañeros, con usuarios, con stakeholders— sigue siendo humana.
- **Responsabilidad:** la IA no puede ser responsable de sus outputs. Cuando algo falla, un humano debe responder. La responsabilidad es intrínsecamente humana.
- **Creatividad profunda:** la IA puede combinar patrones existentes de formas novedosas, pero la creatividad que rompe paradigmas, que imagina lo que no existe, sigue siendo un territorio humano.
- **Liderazgo:** inspirar, motivar, dar dirección en momentos de incertidumbre, sostener a otros en momentos difíciles. El liderazgo genuino es relacional y humano.
- **Ética:** decidir qué está bien y qué está mal, qué es aceptable y qué no, especialmente en situaciones novedosas que no tienen precedente claro.

Competencias que se vuelven más valiosas

Paradójicamente, las competencias tradicionalmente llamadas «blandas» se vuelven las más «duras» en la era de la IA:

- Comunicación efectiva (especialmente escucha y empatía).
- Pensamiento crítico (especialmente escepticismo constructivo).
- Resolución de conflictos.
- Facilitación de grupos.
- Negociación e influencia.
- Visión estratégica.

- Adaptabilidad y aprendizaje continuo.

Estas competencias siempre fueron importantes. La IA las hace críticas.

El desarrollo del valor humano

¿Cómo desarrollar estas competencias?

- **Práctica deliberada:** no basta con «tener experiencias». Hay que practicar deliberadamente las competencias específicas, con feedback.
- **Feedback externo:** las competencias relacionales requieren perspectivas externas. Buscar feedback de colegas, mentores, coaches.
- **Reflexión:** las retrospectivas, los diarios, las conversaciones de desarrollo. Los espacios de reflexión son donde las experiencias se convierten en aprendizaje.
- **Exposición:** buscar situaciones que estiren las competencias. Liderar una reunión difícil, dar feedback incómodo, facilitar un conflicto.
- **Formación:** cursos, talleres, certificaciones en áreas específicas. El aprendizaje formal complementa el aprendizaje experiencial.

8.4. Tu plan de desarrollo profesional

La evolución del equipo y la organización empieza por la evolución individual. Cada profesional ágil necesita un plan de desarrollo personal para la era de la IA.

Evaluación de punto de partida

Antes de planificar, evalúa dónde estás:

Competencias técnicas con IA

- **Prompt engineering:** ¿Sabes comunicarte efectivamente con la IA?
- **Spec writing:** ¿Sabes escribir especificaciones que la IA pueda seguir?
- **Revisión de código de IA:** ¿Sabes qué buscar en código generado?
- **Herramientas:** ¿Conoces las herramientas disponibles y sus capacidades?

Competencias de valor humano

- **Comunicación:** ¿Escuchas bien? ¿Comunicas con claridad?
- **Pensamiento crítico:** ¿Cuestionas constructivamente? ¿Evalúas evidencia?
- **Liderazgo:** ¿Inspiras a otros? ¿Tomas decisiones difíciles?
- **Facilitación:** ¿Ayudas a grupos a funcionar mejor?
- **Visión estratégica:** ¿Ves el panorama completo? ¿Anticipas el futuro?

Para cada competencia, evalúa: fortaleza actual (1-10), importancia para tu rol (1-10), brecha (importancia - fortaleza).

Priorización

No puedes desarrollar todo a la vez. Prioriza basándote en:

- **Brecha:** donde la importancia es alta y la fortaleza es baja.
- **Impacto:** qué competencia tendría más impacto en tu efectividad.
- **Oportunidad:** qué competencia puedes practicar en tu contexto actual.
- **Interés:** qué te motiva aprender (la motivación acelera el aprendizaje).

Selecciona 1-2 competencias para enfocarte durante los próximos 3-6 meses.

Plan de acción

Para cada competencia priorizada:

- **Objetivo específico:** ¿Qué quieres ser capaz de hacer que hoy no puedes?
- **Recursos de aprendizaje:** ¿Qué cursos, libros, mentores puedes usar?
- **Práctica deliberada:** ¿Qué situaciones buscarás para practicar?
- **Feedback:** ¿Quién te dará feedback sobre tu progreso?
- **Métricas de progreso:** ¿Cómo sabrás que estás mejorando?
- **Timeline:** ¿Cuándo revisarás tu progreso?

Integración con el trabajo

El mejor desarrollo ocurre integrado con el trabajo real:

- Ofrécete para facilitar la próxima retrospectiva (práctica de facilitación).
- Pide feedback después de cada reunión que lideres (práctica de liderazgo).
- Revisa críticamente el código de IA antes de aceptarlo (práctica de pensamiento crítico).
- Escribe specs para las próximas funcionalidades (práctica de spec writing).
- Propón un experimento en la próxima retrospectiva (práctica de iniciativa).

Revisión regular

El plan de desarrollo no es estático:

- **Revisión mensual:** ¿Estoy progresando? ¿Qué ajustes necesito?
- **Revisión trimestral:** ¿Siguen siendo estas las prioridades correctas?
- **Conversación con manager/mentor:** compartir el plan y pedir apoyo.
- **Celebrar progreso:** reconocer lo que has logrado, no solo lo que falta.

Ejercicios prácticos

Ejercicio 1: facilitación de sesión de canvas

Prepara una sesión para usar el Canvas de autoevaluación con tu equipo:

PREPARACIÓN (antes de la sesión)

1. **Agenda propuesta:**
    - Check-in: 5 min.
    - Explicación del canvas: 5 min.
    - Evaluación individual: 10 min.
    - Compartir y patrones: 20 min.
    - Priorización: 10 min.
    - Diseño de experimento: 15 min.
    - Check-out: 5 min.
    - Total: ~70 min.
2. **Materiales necesarios:**
    - Canvas impreso o en pizarra.
    - Post-its o herramienta digital equivalente.
    - Rotuladores.
3. **Preguntas para cada dimensión (preparar 2-3 por dimensión):**
    - Cadencia:
    - Composición:
    - Lenguaje:
    - Roles:
    - Calidad:

DURANTE LA SESIÓN

4. ¿Cómo asegurarás que todas las voces se escuchen?
5. ¿Cómo manejarás si surge tensión o desacuerdo?
6. ¿Cómo cerrarás con un experimento concreto y comprometido?

DESPUÉS DE LA SESIÓN

7. ¿Cómo documentarás y comunicarás las conclusiones?
8. ¿Cuándo revisaréis el resultado del experimento?

Ejercicio 2: evaluación de competencias personales

Evalúa tu perfil de competencias para la era de la IA:

COMPETENCIAS TÉCNICAS CON IA

(1 = no tengo la competencia, 10 = la domino)

- Prompt engineering: ___
- Importancia para mi rol: ___
- Brecha: ___
- Spec writing: ___
- Importancia para mi rol: ___
- Brecha: ___
- Revisión de código de IA: ___
- Importancia para mi rol: ___
- Brecha: ___
- Conocimiento de herramientas: ___
- Importancia para mi rol: ___
- Brecha: ___

COMPETENCIAS DE VALOR HUMANO

- Comunicación efectiva: ___
- Importancia para mi rol: ___
- Brecha: ___
- Pensamiento crítico: ___
- Importancia para mi rol: ___
- Brecha: ___
- Liderazgo e influencia: ___
- Importancia para mi rol: ___
- Brecha: ___
- Facilitación: ___
- Importancia para mi rol: ___
- Brecha: ___
- Visión estratégica: ___
- Importancia para mi rol: ___
- Brecha: ___

PRIORIZACIÓN

Las 2 competencias con mayor brecha relevante:

1.
2.

La competencia que más impacto tendría:

La competencia que más me motiva desarrollar:

Ejercicio 3: plan de desarrollo personal

Crea tu plan de desarrollo para los próximos 3 meses:

COMPETENCIA PRIORITARIA:

_____

1. **Objetivo específico:**

   ¿Qué quiero ser capaz de hacer en 3 meses que hoy no puedo?

2. **Recursos de aprendizaje:**
   - Cursos/formación:
   - Libros/artículos:
   - Mentores/coaches:
   - Comunidades/peers:
3. **Práctica deliberada:**
   - ¿Qué situaciones buscaré para practicar?
   - ¿Con qué frecuencia?
   - ¿Cómo integraré la práctica con mi trabajo diario?
4. **Feedback:**
   - ¿A quién pediré feedback?
   - ¿Con qué frecuencia?
   - ¿Qué preguntas específicas haré?
5. **Métricas de progreso:**
   - ¿Cómo sabré que estoy mejorando?
   - ¿Qué indicadores observaré?
6. **Timeline:**
   - Revisión mensual: día ___ de cada mes
   - Hito a los 3 meses: _____
7. **Apoyo necesario**
   - ¿Qué apoyo necesito de mi organización?
   - ¿Cómo lo pediré?

COMPROMISO:

Me comprometo a revisar este plan el día ___ de cada mes.

Compartiré este plan con: _____

Resumen del capítulo

En este capítulo final hemos explorado cómo liderar la evolución:

- El Canvas de autoevaluación proporciona un marco para evaluar cinco dimensiones (cadencia, composición, lenguaje, roles, calidad) y diseñar experimentos basados en las fricciones identificadas.
- Facilitar la conversación de adaptación requiere curiosidad, datos, inclusión de todas las voces y enfoque en experimentación. Las preguntas poderosas abren el diálogo; la gestión de la resistencia lo hace sostenible.
- El valor diferencial humano está en lo que la IA no puede hacer: juicio contextual, empatía genuina, responsabilidad, creatividad profunda, liderazgo y ética. Las competencias «blandas» se vuelven las más «duras».
- El plan de desarrollo personal incluye evaluación de punto de partida, priorización, plan de acción con recursos y práctica deliberada, y revisión regular.
- La evolución de scrum ante la IA no es un destino al que se llega; es un viaje continuo de inspección, adaptación y aprendizaje. Los principios ágiles son la brújula. Las prácticas evolucionan. El valor humano permanece.

Preguntas de autoevaluación

1. ¿Cuáles son las cinco dimensiones del Canvas de autoevaluación para equipos?
2. ¿Qué principios deben guiar la facilitación de una conversación sobre adaptación a la IA?
3. Menciona tres competencias que la IA no puede replicar y por qué son más valiosas ahora.
4. ¿Qué elementos debe incluir un plan de desarrollo personal para la era de la IA?
5. ¿Por qué las competencias tradicionalmente llamadas 'blandas' se vuelven las más 'duras' con la IA?

CIERRE — Evolucionar sin perder el rumbo

La IA generativa no ha matado a scrum. Ha hecho algo más sutil y profundo: ha puesto a prueba si los equipos realmente practican la agilidad o simplemente ejecutan un conjunto de rituales.

Ha separado a quienes aportan juicio estratégico de quienes realizan tareas que la tecnología puede automatizar. Ha acelerado la capacidad de construir hasta el punto de que decidir qué construir se ha convertido en el desafío principal.

Lo que hemos aprendido

A lo largo de esta guía hemos explorado:

- **Los principios ágiles son la brújula.** Empirismo, entrega de valor, adaptación continua y colaboración humana no solo sobreviven a la era de la IA: se vuelven más necesarios que nunca. La brújula no cambia; el terreno sí.
- **Las prácticas concretas deben evolucionar.** Cadencias, tamaño de equipos, artefactos, roles y modelos de calidad necesitan una revisión profunda. No se trata de ajustes cosméticos sino de repensar cómo se implementan los principios en un contexto radicalmente nuevo.
- **La velocidad sin validación es desperdicio.** La paradoja de la productividad demuestra que el cuello de botella ya no es la capacidad de construir, sino la capacidad de decidir qué merece ser construido.
- **No hay un único modelo correcto.** El doble carril, el SDD, el framework de IA como teammate y los modelos híbridos son opciones que cada equipo debe evaluar según su contexto. La experimentación disciplinada es el camino.
- **Scrum evoluciona porque es conocimiento abierto.** A diferencia de los modelos propietarios, scrum no necesita esperar a que un comité lo actualice. Su naturaleza de conocimiento emergente y comunitario es precisamente lo que le permite adaptarse a la era de la IA.

El valor que permanece

La evolución de scrum ante la IA no es solo un cambio de herramientas. Es una transformación que requiere claridad sobre qué es valioso:

- **Los principios:** empirismo, autoorganización, entrega iterativa de valor. Estos no cambian.
- **Las personas:** su juicio, su creatividad, su capacidad de relación, su responsabilidad. La IA amplifica lo que las personas aportan, pero no lo sustituye.
- **La reflexión:** los momentos de inspección y adaptación. Paradójicamente, cuando la velocidad aumenta, la reflexión se vuelve más importante, no menos.

El camino adelante

Esta guía no es un punto de llegada; es un punto de partida. Las prácticas que hemos descrito seguirán evolucionando. Los modelos emergentes madurarán o serán reemplazados. Nuevas herramientas aparecerán.

Lo que permanecerá constante es la necesidad de:

- Observar cómo cambia el contexto.
- Experimentar con nuevas formas de trabajar.
- Inspeccionar los resultados con honestidad.
- Adaptar las prácticas basándose en la evidencia.
- Mantener los principios como guía.

Esto es, en esencia, lo que la comunidad ágil ha hecho siempre. La IA es el próximo capítulo de esa historia. Y cada equipo, cada profesional, tiene la oportunidad de escribir su parte.