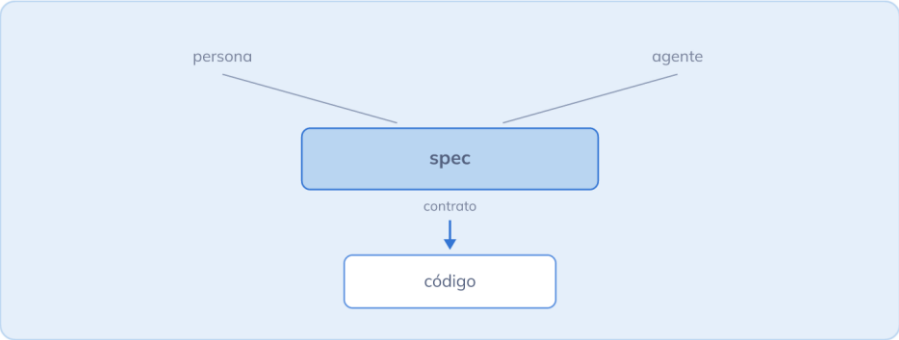
Scrum Manager
Uncovering better ways

State of the art

Spec Driven Development

en equipos ágiles

Mapa de referencia del conocimiento profesional vigente



Actualizado a septiembre de 2026

Scrum Manager® · Skill Arena

Sobre este documento

Este documento es un mapa curado de los conocimientos, prácticas y modelos empleados profesionalmente a la fecha de su publicación para dirigir el desarrollo de *software* con IA mediante especificaciones (*Spec Driven Development*) en equipos ágiles. Para cada uno, este mapa lo sitúa, indica su grado de adopción en la práctica profesional actual y orienta sobre dónde encontrar información de referencia.

Úsalo como observatorio y punto de referencia para contrastar si el conocimiento profesional que empleas está alineado con la práctica actual y en vanguardia.

Se complementa con un documento de desarrollo que profundiza en cada concepto y con la plataforma de entrenamiento y evaluación en Skill Arena. En el área Spec Driven Development en equipos ágiles puedes contrastar tu nivel de conocimiento y, si lo superas, obtener un diploma que acredita curricularmente la solvencia y vanguardia profesional en esta área.

Estado del conocimiento

El conocimiento sobre Spec Driven Development evoluciona de forma extremadamente rápida: método, herramientas y prácticas se renuevan en cuestión de meses. Los fundamentos ya están asentados y el método ha ganado respaldo académico e institucional durante 2026, pero el instrumental sigue en plena formación y la evidencia sobre cuánto acelera realmente el desarrollo con IA está en revisión. En los distintos apartados del documento, las etiquetas (ESTABLECIDO, EN CONSOLIDACIÓN, EMERGENTE) ayudan a identificar la madurez de cada concepto:

ESTABLECIDO consenso asentado; conocimiento que se da por necesario.
EN CONSOLIDACIÓN gana adopción con rapidez; aún no universal pero ya relevante.
EMERGENTE frontera reciente; alta relevancia y alta volatilidad.

Bloque A — El paradigma: por qué SDD

Qué es SDD, qué problema resuelve y por qué no supone un retroceso a waterfall. *Es la base conceptual común a cualquier rol del equipo.*

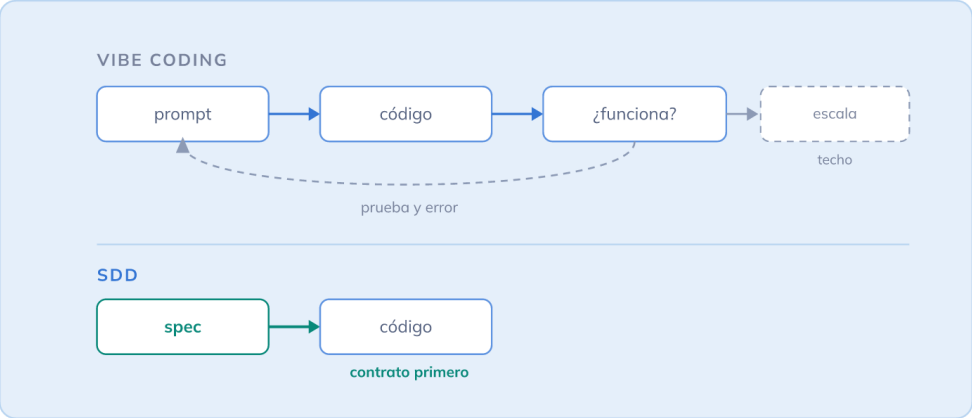


Figura 1. *SDD invierte el orden: el vibe coding conversa con el agente y SDD le entrega un contrato antes de generar código.*

Del *vibe coding* al desarrollo guiado por especificaciones · ESTABLECIDO

El vibe coding consiste en pedir código a la IA de forma conversacional y aceptarlo si funciona. Es extraordinariamente productivo para prototipos, herramientas internas y exploración de ideas, pero tiene un techo que aparece antes de lo que la mayoría espera: a partir de cierta escala genera requisitos implícitos, contaminación de contexto y deriva arquitectónica. El desarrollo guiado por especificaciones (*spec-driven development*) responde invirtiendo el orden, con una *spec* que actúa como contrato antes de generar una línea de código.

Por qué está aquí ahora. La discusión ha madurado y con ella la evidencia, que conviene manejar con cuidado. El ensayo controlado de METR de julio de 2025, con un 19 % más de tiempo empleado con IA frente a un 20 % de aceleración percibida, sigue siendo el dato más citado del área; su propia autora lo rediseñó en febrero de 2026, al comprobar que el seguimiento ya no distinguía el efecto de su ausencia, y hoy cree probable que la aceleración real sea mayor. Lo que sí se sostiene, y el informe de DORA sobre el retorno de la inversión en IA describe con nombre propio, es el impuesto de verificación: el esfuerzo de comprobar que el código generado es fiable y coherente con la arquitectura no desaparece, se traslada.

Dónde mirar. METR — cambio de diseño del experimento (2026) · DORA — ROI of AI-assisted Software Development

La spec como artefacto primario del desarrollo · ESTABLECIDO

El centro del trabajo se desplaza de escribir código a especificar con claridad qué se construye, por qué, bajo qué restricciones y con qué criterios de éxito. La spec es el plano y el código es la consecuencia que se deriva de ella. Para un agente literal, cada decisión que la spec no toma es una decisión que el agente toma por su cuenta.

Por qué está aquí ahora. Es el cambio de identidad profesional que vertebra el tema, y afecta a todos los roles del equipo y no solo a quien programa, porque especificar con precisión es competencia compartida. Durante 2026 la idea ha ganado formulación académica: el trabajo de Díaz, Gayoso, Cimmino y Pérez describe las especificaciones como el sustrato contractual entre humanos y agentes, y sostiene que es lo que devuelve la responsabilidad y la verificabilidad que la asistencia informal de la IA erosiona.

Dónde mirar. Addy Osmani — How to write a good spec for AI agents · SDD for Agentic Software Engineering (arXiv, 2026)

SDD y agilidad: no es el regreso de waterfall · EN CONSOLIDACIÓN

SDD usa fases secuenciales y puertas de aprobación, lo que recuerda a waterfall, pero difiere en tres ejes. El alcance es cada incremento y no el proyecto entero, el *feedback* llega en cada puerta y no al final, y la reversibilidad es alta porque modificar un documento resulta trivial frente a reescribir código. Bien entendido, es la aplicación coherente de los principios ágiles cuando parte del equipo son agentes.

Por qué está aquí ahora. Es la objeción que cualquier profesional ágil plantea ante SDD, y resolverla es lo que permite adoptarlo sin traicionar la agilidad. La crítica tiene nombres y sigue viva: Zaninotto mantiene que SDD revive la documentación pesada previa al código y propone como alternativa el desarrollo en lenguaje natural, y Yeret ha discutido desde el mundo ágil si supone un paso adelante o atrás. El Technology Radar de Thoughtworks lo mantiene en «Assess», sin haberlo promovido a «Trial»: aporta valor con requisitos ambiguos, equipos distribuidos o necesidad de trazabilidad, y puede no compensar en problemas triviales.

Dónde mirar. Thoughtworks Technology Radar — SDD · Marmelab — SDD: The Waterfall Strikes Back

El principio de proporcionalidad · ESTABLECIDO

Consiste en ajustar la intensidad del proceso al tamaño del problema. Un cambio trivial no necesita una spec de cuatro fases; una funcionalidad compleja sí. Mal calibrado, SDD se convierte en una burocracia de especificación que ralentiza en lugar de acelerar.

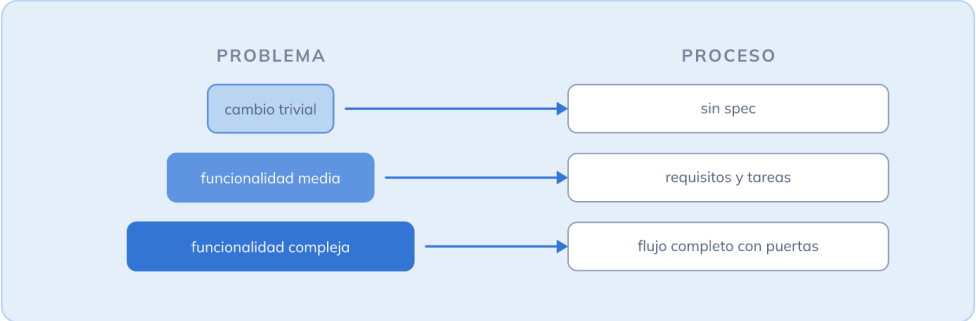


Figura 2. *El principio de proporcionalidad: la intensidad del proceso se ajusta al tamaño del problema.*

Por qué está aquí ahora. Es el criterio que separa el uso útil de SDD de su caricatura burocrática. Böckeler documentó cómo una herramienta convirtió un *bug* menor en cuatro historias con dieciséis criterios de aceptación, y lo describió como usar un mazo para romper una nuez. La evidencia acumulada durante 2026 refuerza el principio desde otro ángulo: el retorno de la IA es alto en trabajo nuevo y sencillo y mucho menor en código heredado y complejo, de modo que la intensidad del proceso no puede ser la misma en los dos casos. Saber cuándo no aplicar SDD forma parte de dominarlo.

Dónde mirar. Böckeler — Understanding Spec-Driven Development · Scrum Manager — Scrum en la era de la IA

Bloque B — El método: fases, envolturas y puertas

La mecánica de SDD: el flujo de trabajo, los artefactos que lo enmarcan y las puertas de aprobación que lo hacen viable. Es el núcleo operativo del tema.

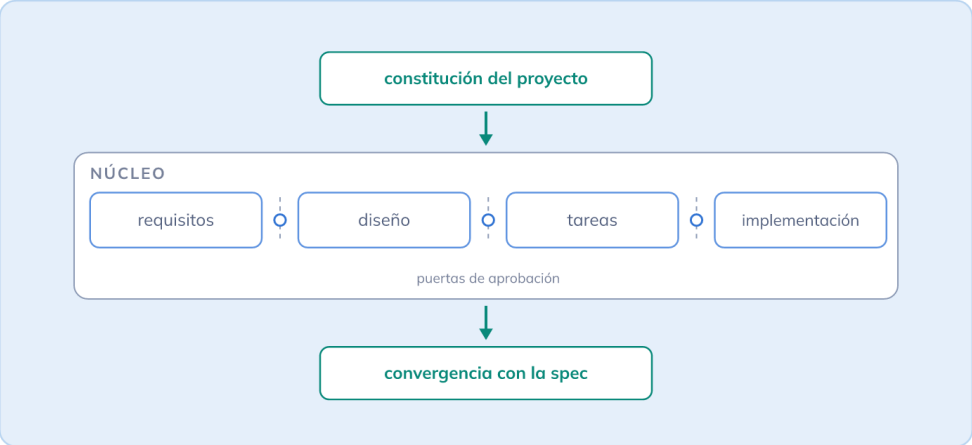


Figura 3. *El núcleo de cuatro fases y sus dos envolturas: la constitución precede a los requisitos, la convergencia cierra la implementación y la revisión se concentra en las puertas.*

El flujo: el núcleo de cuatro fases y sus dos envolturas · ESTABLECIDO

El núcleo del método es una secuencia de cuatro fases, cada una con su documento: requisitos (qué se construye), diseño (cómo), tareas (en qué unidades atómicas) e implementación (la ejecución). Alrededor de ese núcleo se han asentado dos envolturas. Antes de los requisitos se establecen los principios del proyecto en una constitución, y después de la implementación se comprueba que el resultado converge con lo especificado.

Por qué está aquí ahora. El núcleo de cuatro fases es la columna vertebral estable del método y refleja la secuencia lógica mínima para pasar de una intención a código cuando el constructor es un agente. Las envolturas son la novedad que obliga a reformular la ficha: GitHub Spec Kit, la implementación más adoptada, documenta hoy seis pasos que empiezan por la constitución y terminan convergiendo la implementación con las especificaciones, con comandos añadidos de clarificación, análisis y listas de comprobación. Kiro llega a lo mismo por otro camino, con sus flujos de requisitos primero o diseño primero y un análisis profundo opcional de los requisitos antes de pasar al diseño.

Dónde mirar. GitHub Spec Kit · Kiro — buenas prácticas de specs

La constitución del proyecto · EN CONSOLIDACIÓN · *nuevo desde 2026*

La constitución recoge lo que se decide una vez para todo el proyecto, frente a lo que se decide en cada incremento. Es el reglamento fundacional que alinea el ciclo de vida: alcance del proyecto, contexto de dominio, versiones tecnológicas, estándares de código

y estructura del repositorio. Persiste por encima de las specs individuales y es lo que da continuidad al criterio entre sesiones distintas del agente.

Por qué está aquí ahora. Ha pasado de práctica implícita a artefacto con nombre y con su propio paso en el método: en Spec Kit es el primero de los seis y precede a los requisitos. Los equipos de Thoughtworks que trabajan con SDD en bases de código existentes informan de qué contiene una constitución que funciona de verdad, y la formación del área ya la enseña como primera competencia, antes que las specs de funcionalidad, porque es lo que preserva el contexto entre sesiones y reduce la deuda cognitiva.

Dónde mirar. Thoughtworks Technology Radar — GitHub Spec Kit · DeepLearning.AI — Spec-Driven Development with Coding Agents

Las puertas de aprobación (*approval gates*) · ESTABLECIDO

Son los puntos donde la persona ejerce criterio entre fases y responde a tres preguntas: si los requisitos son correctos, si el diseño resulta viable y si las tareas cubren el diseño. Concentran la revisión donde la información vale más y el coste de corrección es menor, en lugar de supervisar cada acción del agente. El principio rector se resume en revisar en las puertas, no durante la implementación.

Por qué está aquí ahora. Es lo que hace viable delegar la construcción sin perder el control, y evita la fatiga de aprobación de ir validando microdecisiones una a una. La evidencia de 2026 añade un contrapunto que conviene conocer: DORA identifica la revisión manual como el cuello de botella que se satura cuando la velocidad de generación de código sube, de modo que una puerta mal dimensionada deja de proteger y empieza a frenar. Morris ha explorado en la misma línea cómo se reparten personas y agentes dentro de los bucles de la ingeniería de software.

Dónde mirar. Kief Morris — Humans and Agents in Software Engineering Loops · DORA — ROI of AI-assisted Software Development

Tareas atómicas, oleadas y *commits* atómicos · EN CONSOLIDACIÓN

El diseño se descompone en unidades de implementación pequeñas y verificables por separado, cada una con su *prompt* estructurado. Las tareas independientes se agrupan en oleadas que pueden ejecutarse en paralelo, y cada una cierra con su propio commit. Así el trabajo del agente queda revisable pieza a pieza y es reversible sin arrastrar lo demás.

Por qué está aquí ahora. Es la unidad de trabajo que hace que la fase de implementación se pueda supervisar sin leer todo el código de una vez, y la que permite descartar una tarea mal resuelta sin deshacer el incremento entero. La granularidad sigue siendo materia de criterio profesional más que de consenso cerrado, y de ahí su estado.

Dónde mirar. Anthropic — Building effective agents · GitHub Spec Kit

Especificar sobre *codebase* existente (*brownfield*) · EN CONSOLIDACIÓN

En un proyecto con historia, la spec no describe el sistema entero: describe el delta. Antes de escribir un requisito conviene responder qué ficheros se verán afectados, qué patrones y dependencias existen ya y qué no debe romperse, y después fijar lo que existe para que el agente no lo sobrescriba. Una especificación completa del sistema es impracticable a escala, y además innecesaria.

Por qué está aquí ahora. Es donde se juega la utilidad real del método, porque la mayoría del trabajo profesional ocurre sobre código que ya existe. Los equipos de Thoughtworks informan de que usan SDD sobre todo en entornos brownfield, precisamente donde la intención poco clara y los supuestos ocultos generan retrabajo; Kiro ofrece generar specs de lo que ya está construido, y OpenSpec se declara directamente brownfield primero. Nota terminológica: la versión anterior de este mapa llamaba «Impact Report» a este análisis previo, un nombre que no se usa en el área ni en la documentación de las herramientas, y que esta edición retira.

Dónde mirar. OpenSpec · Kiro — buenas prácticas de specs

Bloque C — Escribir buenas specs

La competencia central del tema: producir especificaciones que un agente pueda ejecutar sin adivinar. Aquí está la parte del trabajo que no delega ninguna herramienta.

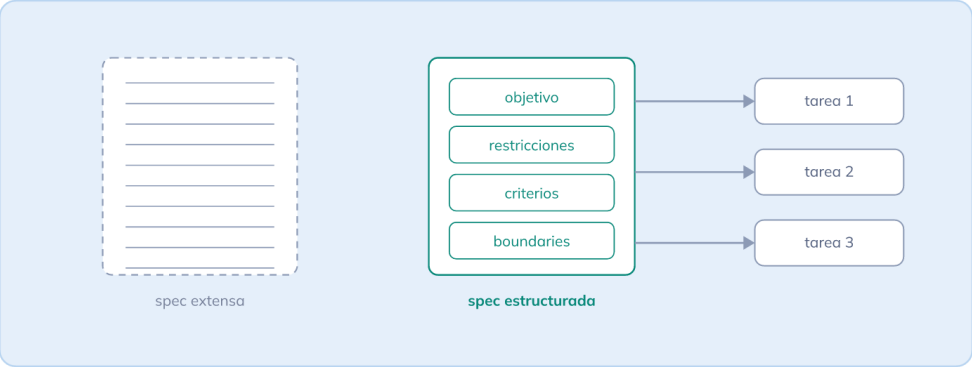


Figura 4. *La spec inteligente, no extensa: cada tarea recibe la porción de la spec que le corresponde, no el documento completo.*

La spec inteligente, no extensa · ESTABLECIDO

Una buena spec no se mide por su longitud. Los principios que se han asentado son partir de la visión de alto nivel y dejar que la IA elabore el detalle, estructurar el documento como material profesional, alimentar cada tarea con la porción relevante y no con la spec completa, incorporar autoverificación y restricciones, y tratarla como documento vivo.

Por qué está aquí ahora. Es la competencia que más diferencia el resultado, y la que menos depende de la herramienta elegida. Osmani fijó los principios de referencia y ha seguido desarrollándolos en su trabajo sobre ingeniería agéntica. En abril de 2026, Zhang y Xia añadieron una vuelta de tuerca con el desarrollo guiado por prompts estructurados, que trata el prompt como artefacto versionado y gobernado en lugar de algo que se genera una vez y se descarta, y aporta una plantilla de siete partes para construirlo.

Dónde mirar. Addy Osmani — Agentic engineering · Zhang y Xia — Structured-Prompt-Driven Development

La notación EARS para criterios de aceptación · EN CONSOLIDACIÓN

EARS (*Easy Approach to Requirements Syntax*) estructura el lenguaje natural lo justo para eliminar la ambigüedad sin perder legibilidad. Su plantilla básica encaja precondición, evento y respuesta esperada del sistema, y define cinco patrones que cubren la mayoría de los requisitos. Un criterio en EARS es verificable por construcción.

Por qué está aquí ahora. Los agentes responden bien a requisitos estructurados con patrones consistentes, porque la forma fija reduce el margen de interpretación. La notación nació en el sector aeroespacial hace más de una década y ha vuelto al primer plano al integrarse en el instrumental de SDD, señaladamente en Kiro. Su estado refleja

que la adopción crece con rapidez sin ser todavía universal, y que se usa donde reduce ambigüedad de verdad y no como obligación.

Dónde mirar. Alistair Mavin — EARS · Kiro — buenas prácticas de specs

El sistema de *boundaries*: *Always / Ask First / Never* · EN CONSOLIDACIÓN

Los boundaries declaran qué puede hacer el agente por su cuenta, qué debe consultar antes y qué no debe hacer nunca. Operan en dos niveles: en la constitución del proyecto fijan reglas generales, y en cada tarea añaden restricciones concretas. Son la diferencia entre delegar y desatender.

Por qué está aquí ahora. Es el mecanismo que hace segura la autonomía del agente en la fase de implementación, y el que permite subir esa autonomía sin subir el riesgo. El soporte del ecosistema ha dejado de ser cosa de cada herramienta: AGENTS.md se ha convertido en el formato más parecido a un estándar universal de instrucciones para agentes, lo leen de forma nativa una veintena de herramientas y lo han adoptado más de sesenta mil proyectos, y su especificación está tutelada por la Agentic AI Foundation de la Linux Foundation.

Dónde mirar. AGENTS.md · Agentic AI Foundation

La maldición de las instrucciones (*instruction overload*) · ESTABLECIDO

Añadir instrucciones a un agente no mejora su comportamiento de forma indefinida. Pasado cierto punto, más contexto degrada la capacidad de recuperar lo importante, y lo que queda en el medio de una ventana larga es lo que peor se recuerda. La contramedida es dar a cada tarea la porción de spec que necesita y sistematizar el contexto reutilizable en ficheros dedicados en lugar de engordar un único documento.

Por qué está aquí ahora. El fenómeno ha pasado de observación práctica a disciplina con nombre. La ingeniería de contexto se distingue ya de la ingeniería de prompts, porque la pregunta no es cómo redactar la instrucción sino qué debe ocupar la ventana en cada momento y qué conviene desalojar, comprimir o delegar. Tiene además investigación propia sobre la degradación del contexto y lo que se pierde en el medio. En la práctica de equipo, el conocimiento institucional reutilizable (convenciones de la casa, despliegue, patrones propios) se encapsula en piezas que el agente carga cuando las necesita. Ese doble asentamiento, del nombre y de la evidencia, es lo que sube la ficha a establecido en esta edición.

Dónde mirar. Anthropic — Effective context engineering for AI agents · Thoughtworks Technology Radar — GitHub Spec Kit

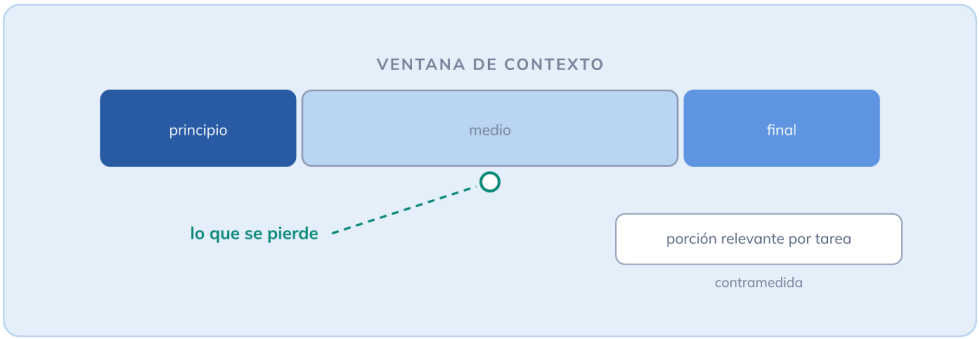


Figura 5. *La maldición de las instrucciones: lo que queda en el medio de una ventana de contexto larga es lo que peor se recuerda, y la contramedida es dar a cada tarea su porción relevante.*

Specs vivas, deriva y la *Clarity Gate* · EN CONSOLIDACIÓN

El riesgo principal de una spec es la deriva respecto al código que la implementa, porque una fuente de verdad desactualizada resulta peor que no tenerla. Frente a eso conviene separar lo que es verdad vigente de lo que solo es propuesta de cambio, y archivar lo ya completado. La Clarity Gate es la prueba de calidad que lo comprueba: si un agente distinto, o una sesión nueva del mismo agente, no puede generar código equivalente leyendo solo la spec, la spec está incompleta.

Por qué está aquí ahora. Es el problema que decide si SDD aporta valor a los seis meses o solo la primera semana. La señal que este mapa anticipaba en su edición anterior se ha resuelto en sentido negativo: el nivel donde la spec es la única fuente y el código se regenera por completo sigue siendo experimental, con el motor de Tessl en beta cerrada y la propia compañía desplazando su peso hacia el registro de especificaciones y la gobernanza. Mientras tanto ha aparecido un patrón práctico y sencillo, el de OpenSpec, que mantiene separados el directorio de la verdad vigente y el de los cambios propuestos.

Dónde mirar. OpenSpec · Tessl

Bloque D — SDD en el equipo y su ecosistema

Cómo encaja el método en un equipo ágil real, qué obligaciones lo rodean, en qué falla cuando falla y con qué se implementa hoy.

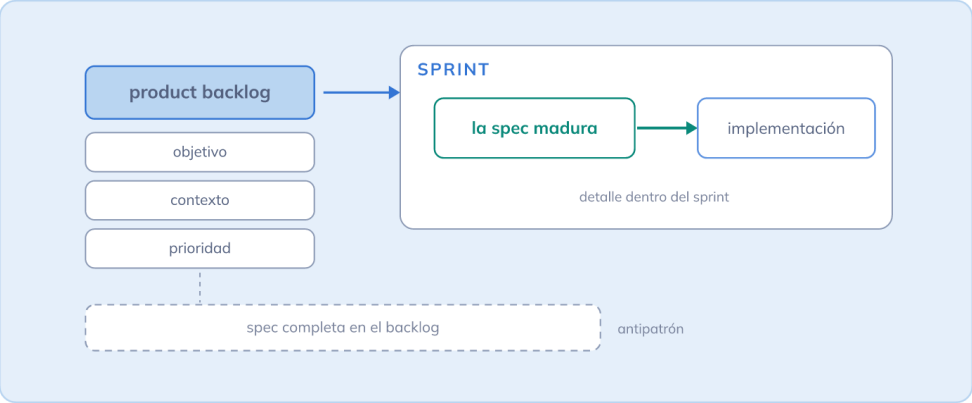


Figura 6. *Dónde vive la spec: el product backlog acuerda objetivo, contexto y prioridad, y el contrato madura dentro del sprint.*

Encaje en roles y ceremonias ágiles · EN CONSOLIDACIÓN

La spec no sustituye al *backlog* ni al refinamiento. En el backlog y el refinamiento el equipo acuerda objetivos, contexto y prioridad; el contrato detallado madura dentro del *sprint*, como parte de hacer el trabajo. El error de integración más habitual consiste en llevar la spec completa al *product backlog*, que la convierte en un documento que caduca antes de ejecutarse.

Por qué está aquí ahora. Es la pregunta práctica que se hace todo equipo que ya trabaja con *scrum* o *kanban*, y donde ha aparecido doctrina utilizable durante 2026. El trabajo académico del área describe además la tensión que el equipo tiene que gestionar: la eficiencia individual sube mientras la capacidad de revisión del equipo se degrada y la estabilidad baja, y propone las especificaciones como gobierno del comportamiento de los agentes a escala de equipo, con un doble *harness* de controles técnicos alrededor del agente y gobernanza metodológica alrededor del equipo.

Dónde mirar. SDD for Agentic Software Engineering (arXiv, 2026) · Yuval Yeret — ¿paso adelante o atrás?

Trazabilidad y supervisión humana como obligación · EMERGENTE · *nuevo desde 2026*

Desde el 2 de agosto de 2026 se aplica el marco de alto riesgo del Reglamento europeo de IA, que exige registro automático de las operaciones del sistema, supervisión humana asignada a personas con la competencia y la autoridad necesarias, y transparencia sobre el contenido generado. Conviene precisar el alcance: lo que el Reglamento regula es el sistema de IA de alto riesgo que el equipo construye, no el uso de IA para programarlo.

Por qué está aquí ahora. Aparece en el mapa porque cambia el valor de dos piezas que el método ya tenía. Cuando el producto entra en el ámbito de alto riesgo, la cadena que

va del requisito al código y las puertas de aprobación documentadas dejan de ser solo buenas prácticas de ingeniería y pasan a ser material aprovechable para el expediente de conformidad. La relación es de utilidad y no de obligación directa, y su estado refleja que la práctica profesional alrededor de esta intersección está aún por hacerse.

Dónde mirar. Reglamento europeo de IA — art. 12, registros · art. 26, obligaciones del responsable del despliegue

Antipatrones: teatro de especificación y documentación zombi · EN CONSOLIDACIÓN

Los modos de fallo característicos de SDD mal aplicado son cuatro. La sobreespecificación invierte más tiempo en la spec que en el software; las puertas se convierten en cuellos de botella; el teatro de especificación produce specs que nadie revisa de verdad; y la documentación zombi deja specs que nadie consulta ni actualiza. Reconocerlos forma parte del dominio del método.

Por qué está aquí ahora. Son los puntos donde SDD deja de aportar valor y empieza a estorbar, y aparecen con facilidad si no se calibra con el principio de proporcionalidad. La crítica externa los describe sin piedad y merece leerse de primera mano: documentos enormes, pérdida del contexto del código, trabajo de revisión duplicado y una falsa sensación de seguridad. Böckeler ha planteado la misma pregunta sobre el bucle del agente al examinar si el desarrollo guiado por pruebas dentro de ese bucle aporta valor real o solo teatro, y el paralelismo es instructivo.

Dónde mirar. Marmelab — SDD: The Waterfall Strikes Back · Böckeler — TDD inside the agent loop

El ecosistema de herramientas · EMERGENTE

El instrumental crece deprisa y la lista concreta caduca antes que el método, de modo que lo que conviene aprender son los criterios de elección. Los que separan hoy unas herramientas de otras son cuatro: si el flujo es ligero o completo, si el trabajo es sobre proyecto nuevo o sobre código existente, si se prefiere una CLI dentro del repositorio o un entorno integrado, y cuánta gobernanza y trazabilidad se necesita. La advertencia de fondo se mantiene: SDD es el método, no la herramienta.

Por qué está aquí ahora. Es la zona más volátil del tema y también la que da la señal más clara de que el método se está institucionalizando. La implementación de referencia publicó su versión 1.0 en agosto de 2026, al cumplir un año, con un razonamiento revelador: renuncia a prometer estabilidad y reivindica adaptabilidad. Alrededor han cuajado opciones con perfiles distintos, desde entornos integrados hasta herramientas deliberadamente ligeras orientadas a código existente, y el propio método tiene ya entrada en el Technology Radar, formación reglada y literatura académica. Conviene contrastar cualquier fotografía del panorama, incluida esta, con las comparativas que se mantienen al día.

Dónde mirar. GitHub Spec Kit — releases · Böckeler — Understanding Spec-Driven Development

Nota sobre las referencias

Los enlaces incluidos estaban disponibles y verificados en la fecha de actualización. Dado el ritmo de cambio del área, algunos pueden modificarse; cada revisión del mapa actualiza también sus referencias.