BIG school {m}

CURSO DE DESARROLLO CON IA

Programa con agentes

Índice

Spec-Driven Development (SDD)

¿Qué es?

El **Spec-Driven Development (SDD)** no es solo una metodología, es una estrategia de mitigación de riesgos frente al indeterminismo de la IA. En un entorno profesional, la especificación técnica actúa como el "ancla de verdad" y la memoria persistente del proyecto.

Al definir requerimientos técnicos precisos antes de la generación de código, transformamos la interacción con la IA de un ejercicio de "adivinación" a uno de implementación guiada, asegurando que el modelo opere dentro de un espacio de soluciones validado.

El SDD es la metodología definitiva que devuelve el control al ingeniero. Desplaza el foco de la "escritura de código" hacia la "validación de la intención". En este flujo, el humano actúa como el Validador de Intentos, mientras que el código se convierte en un artefacto transitorio y derivado.

Consiste en dejar de pedirle al agente “hazme X” y esperar que acierte. Primero creas una especificación. La spec es el cerebro/contexto persistente.

El objetivo es reducir:

- **Pérdida de contexto entre sesiones**
- **Cambios de rumbo invisibles**
- **Ambigüedad/Indeterminismo** → La especificación sirve como el contrato técnico que garantiza que el output de la IA sea consistente a lo largo de diferentes sesiones de desarrollo
- **Vibe coding de prueba/error** → Se sustituye el ciclo ineficiente de prompts iterativos por una implementación precisa desde el primer intento.
- **Deuda cognitiva** → El ingeniero se libera de la carga de supervisar errores triviales, delegando la ejecución táctica a la IA bajo un marco de reglas estricto.

El problema que resuelve

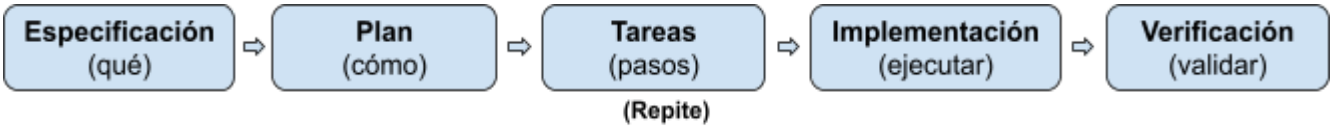
- **Sin SDD**: Desarrollo rápido, pero se produce código que no siempre coincide con lo que buscas. Si las instrucciones son ambiguas, el resultado es ambiguo.

- **Con SDD**: Escribes una especificación clara que define qué construir, y dejas que el agente la implemente. La spec actúa como un guardarraíl.

El ciclo SDD

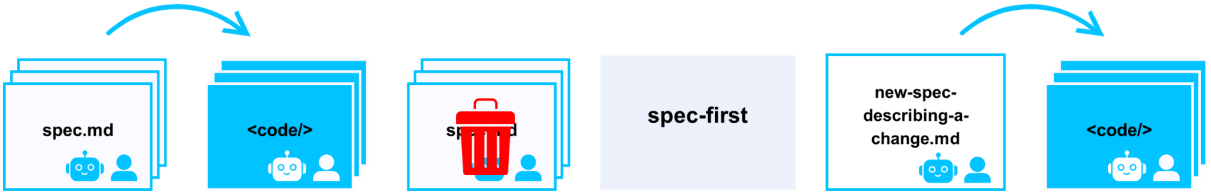
1. **Constitución (una vez por proyecto)**: las reglas generales.
2. **Specify**: qué se va a construir (la feature), con criterios de aceptación.
3. **Plan**: cómo se va a construir (enfoque técnico, archivos, datos).
4. **Tasks**: el plan dividido en tareas pequeñas y verificables.
5. **Implement**: el agente ejecuta las tareas, una a una.
6. **Verify**: se valida contra los criterios de aceptación. Si falla, se ajusta.

El bucle SDD

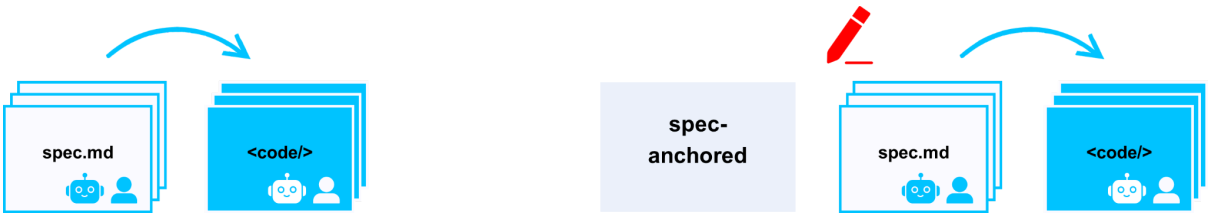


Niveles de SDD

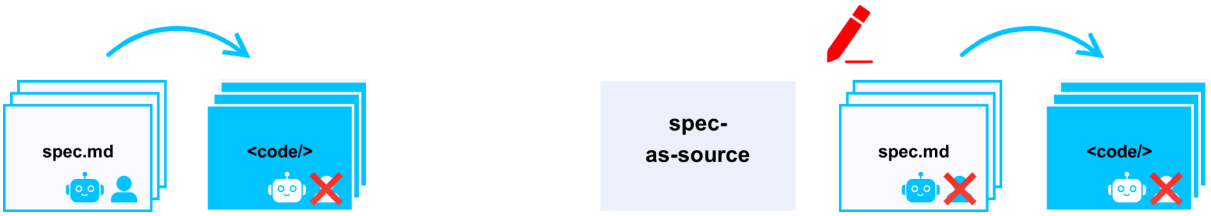
- **Spec-first**: escribes la spec, generas el código una vez, y a partir de ahí editas el código directamente. La spec fue el punto de partida.



- **Spec-anchored**: la spec se mantiene viva: cada cambio importante pasa primero por actualizar la spec. Es el más recomendable para empezar.



- **Spec-as-source**: la spec es lo único que editas; el código se regenera siempre desde ella. Experimental, todavía poco común.



Los artefactos

mi-proyecto/
├── **spec**/ (estructura base)
│ ├── **constitution**/ (puede ser un único archivo)
│ │ ├── **mission.md** ← **qué construimos y para quién**
│ │ ├── **tech-stack.md** ← **tecnologías y convenciones**
│ │ └── **roadmap.md** ← **orden de las features**
│ └── **features**/ (y para cada feature)
│ ├── **001-tarea**/ (puede ser un único archivo)
│ │ ├── **spec.md** ← **qué hace esta feature y criterios de aceptación**
│ │ ├── **plan.md** ← **cómo se implementa**
│ │ └── **tasks.md** ← **checklist de tareas**
│ └── **002-tarea**/
│ └── ...
└── **código**/ ← **el código que genera el agente**

Descarga aquí la plantilla genérica para documentar cualquier proyecto con desarrollo dirigido por especificación (SDD).

Multiagentes

Para abordar la complejidad técnica sin desbordar el contexto del modelo, debemos aplicar el principio de separación de preocupaciones (Separation of Concerns) mediante ecosistemas multiagente. Transicionar de un agente generalista a una red de agentes especializados permite manejar tareas concurrentes con mayor precisión.

Subagente

Un subagente es un agente "hijo" que el agente principal lanza para una tarea concreta. La clave: corre en su propia ventana de contexto, con su propio rol y sus propios permisos, y al terminar devuelve solo un resumen al agente principal.

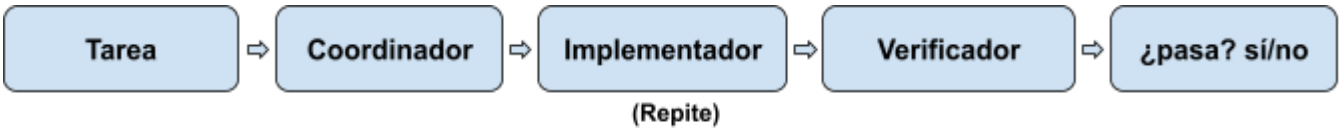
Objetivos de los Subagentes

Aislar contexto: El subagente hace el trabajo "duro" (leer cuarenta archivos, ejecutar tests, rebuscar en el código) y te devuelve solo la conclusión. El agente principal no se llena de basura.

Especializar con permisos mínimos: Cada subagente puede tener un rol concreto y restringido. Un subagente para investigar, otro para auditar, otro para implementar...

El flujo multiagente

- **Coordinador**: No escribe código. Lee la tarea, la divide y reparte el trabajo. Decide qué hace cada subagente y en qué orden.
- **Implementador**: Recibe una subtarea concreta y la ejecuta (escribe el código). Puede haber varios en paralelo, cada uno con su parte.
- **Verificador**: Revisa lo que produjo el implementador. Corre tests, busca fallos, comprueba que cumple lo pedido.



Loop engineering

La Ingeniería de Loops es el diseño y optimización de bucles autónomos de retroalimentación donde el sistema (IA) itera de forma independiente hasta satisfacer las condiciones de salida.

En este paradigma, el ciclo de SDD se automatiza: el agente implementa, el subagente de QA verifica y, si existen discrepancias, el bucle se reinicia automáticamente para corregir el código.

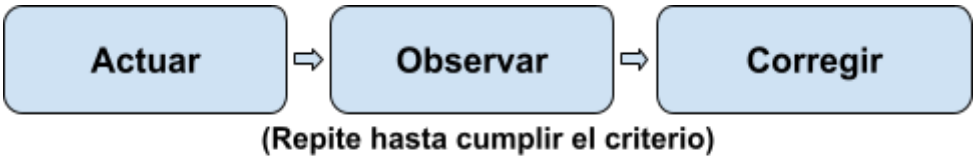
Se diseña un bucle donde el agente itera solo hasta cumplir un objetivo.

Ejemplo:

"Arregla todos los tests" → el agente prueba, ve qué falla, corrige y vuelve a probar… hasta que todo funciona.

Dónde encaja: Prompt engineering (qué pides) → Context engineering (qué sabe) → Harness engineering (su entorno) → Loop engineering (cómo itera)

El bucle



Las claves

- Del prompt → al agente con arnés → a la orquestación con specs.
- La **habilidad que gana valor**: fundamentos, requisitos, diseño y evaluación crítica del código y el proyecto.
  La que pierde: sintaxis pura.
- **Regla de oro: la IA ejecuta, pero el responsable del proyecto eres tú**.

LA POTENCIA SIN CONTROL NO SIRVE DE NADA

Conviértete en Desarrollador con IA

MATRICÚLATE AQUÍ