BIG school

Apuntes

Curso de Desarrollo con IA

DE 0 A PRODUCCIÓN

DÍA 2

Índice

Editores e IDEs en la era de la IA

Desde la irrupción masiva de la inteligencia artificial a finales de 2022, el entorno de desarrollo integrado (IDE) ha dejado de ser una simple superficie de escritura para transformarse en un socio cognitivo. En este nuevo paradigma, el editor no solo asiste en la sintaxis, sino que participa en la lógica y la toma de decisiones arquitectónicas. Como arquitectos, debemos entender que no estamos ante una mejora incremental, sino ante un cambio en la naturaleza misma de la interacción hombre-máquina.

Evolución de los editores de código:

2022 - IDEs clásicos (autocompletado)
2023 - Chat-based coding
2024 - Copilots
2025 - AI agents
2026 - AI-first development environments


VS Code


Claude Code


Codex


Cursor


Antigravity


Warp


V0

Otros IDEs: Zed - Trae - Kiro - Replit - Lovable - IntelliJ - Neovim...

Agentes de código

Para un profesional de la ingeniería de software, seguir utilizando un chat convencional para programar es una ineficiencia estratégica. El salto cualitativo hacia los flujos agénticos permite que el desarrollador opere en un nivel de abstracción superior, validando intenciones en lugar de corregir líneas de código aisladas.

Chatbot

- Pregunta → Respuesta
- Sin acceso al sistema
- Un solo turno
- Requiere copy-paste

Agente

- Bucle autónomo
- Lee/escribe archivos
- Ejecuta y verifica
- Itera

El bucle del Agente

AI-First IDE

El paradigma **AI-First** prioriza herramientas donde el LLM es el núcleo de la arquitectura del editor, no un plugin añadido.

1. Contexto completo:
El modelo ve todo tu proyecto, no sólo el archivo actual.

2. Edición multi-archivo:
Cambios coordinados en múltiples ficheros a la vez.

3. Ejecución real:
Comandos, test y builds dentro del proyecto.

4. Iteración autónoma:
Detectan errores y se auto-corrigen.

EL DESARROLLADOR PASA
A SER EL DIRECTOR DEL PROCESO

Plan Mode

¿Qué es?

El agente primero elabora un plan detallado de los cambios que va a hacer, te lo muestra para que lo revises, y sólo ejecuta cuando tú das el OK. Evita cambios destructivos y permite ajustar la estrategia antes de tocar el código.

Cuándo usarlo

- ✓ Tareas grandes
- ✓ Refactors
- ✓ Supervisión de estrategia

Cuándo NO usarlo

- ✗ Tareas pequeñas
- ✗ Correcciones obvias
- ✗ Prototipado rápido

Reglas y AGENT.md

¿Qué son?

Archivos de texto que actúan como system prompts persistentes para tu proyecto. Para evitar el ruido y la ineficiencia, el agent.md no debe exceder las 500 líneas.

AGENT.md → Genérico	CLAUDE.md	.cursorrules	copilot-instructions.md

¿Qué incluir?

Stack tecnológico	Convenciones de código	Patrones	Prohibiciones
Estructuras de proyecto	Flujo de trabajo	Testing, CI/CD	Estilo de commits y PRs

https://agents.md

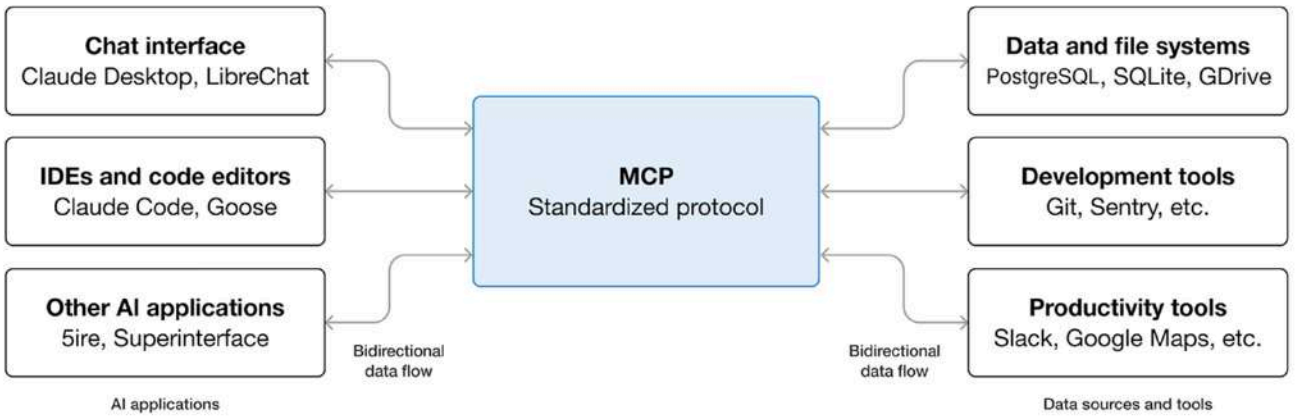
MCP - Model Context Protocol

Cliente MCP

Interfaz estándar para conectar cualquier herramienta

Ventajas

- ✓ Estándar abierto
- ✓ Reutilizable entre IDEs
- ✓ Seguridad por diseño
- ✓ Ecosistema creciente



https://modelcontextprotocol.io

Skills: Conocimiento para agentes

Cliente MCP

- Acceso a herramientas
- Conexión en tiempo real
- Acciones sobre servicios externos
- API estándar (JSON-RPC)

Skills

- Instrucciones/mejores prácticas
- El agente las lee antes de actuar
- Archivos SKILL.md + scripts
- No siempre se usa

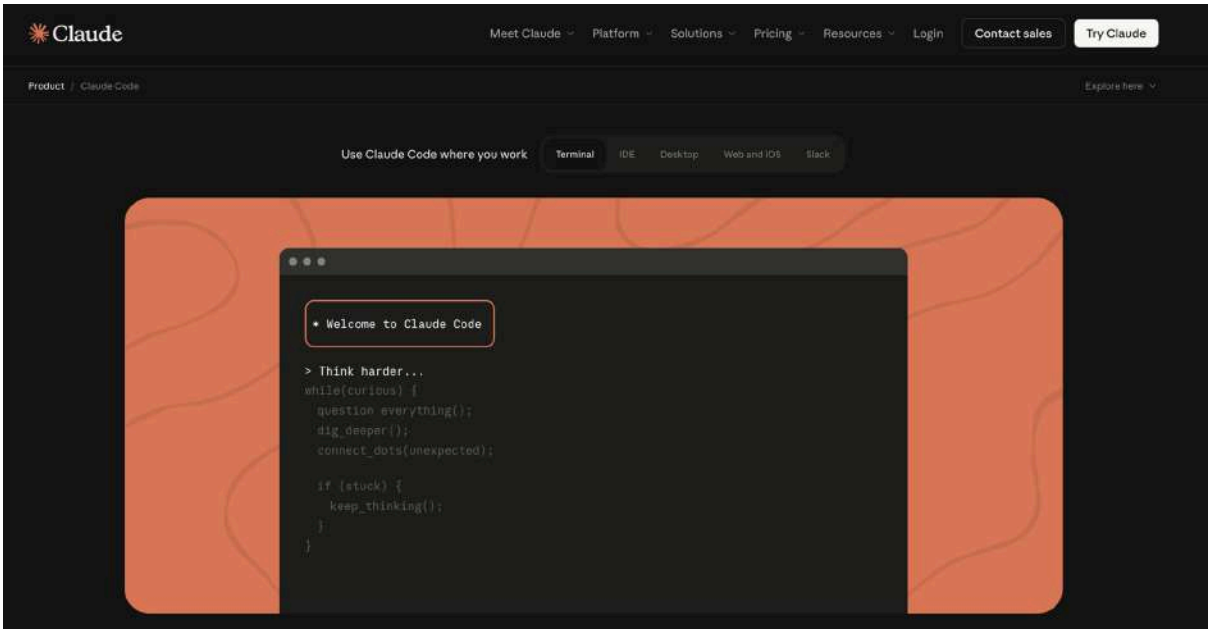
MCPs: Servicios externos

Skills: Conocimiento interno

Utilizando plataformas como skills.sh, podemos dotar al agente de habilidades ultra-específicas (ej. Frontend Design Expert) de forma modular. Esto mantiene el contexto principal limpio y asegura que el agente solo cargue las reglas de diseño React cuando realmente está trabajando en un componente de frontend, optimizando así el uso de tokens y la precisión de la respuesta.

Claude Code

¿Qué es?
Herramienta de codificación agéntica de Anthropic que vive en tu terminal.



https://claude.com/product/claude-code

A continuación, se resumen sus características y funcionamiento principal:

Interfaces de uso: Se puede utilizar de tres maneras principales: desde la terminal (CLI) como una terminal "vitaminada" (ejecutando el comando /claude), mediante una aplicación de escritorio propia, o integrado como una pestaña dentro de editores como Visual Studio Code.

Modelos disponibles: Permite elegir entre diferentes modelos según la complejidad de la tarea y el coste: Claude Opus (el más avanzado y costoso por defecto), Sonnet y Haiku.

Capacidades avanzadas:

Modo Plan (Plan Mode): Diseñado para tareas complejas que requieren una planificación previa antes de ejecutar cambios.

Asincronía: Es capaz de realizar tareas pesadas en segundo plano (background), permitiendo al desarrollador seguir interactuando con la IA mientras esta trabaja.

Agent Teams (Equipos de Agentes): Una funcionalidad de pago que permite al agente principal dividir una tarea y delegarla en subagentes que se comunican entre sí para resolver problemas en paralelo.

Integración de herramientas: Soporta de forma nativa el uso de "Skills" (habilidades y reglas de comportamiento) y MCP (Model Context Protocol) para conectarse con herramientas de terceros y sistemas externos.

Su versión oficial es de pago, aunque existen otras alternativas de código abierto como **Open Code**, que replica y mejora muchas de sus funciones agénticas.

El problema del "Contexto"

La mayoría de los desarrolladores usan la IA como un chat glorificado.

Esto no es un problema del modelo — es un problema de ARQUITECTURA.

Evolución histórica

Cada salto fue impulsado por dolor real, no por tendencia.

Herramienta base

Claude Code
by *Anthropic*

- Agente de terminal con acceso nativo al filestystem.
- Edición, búsqueda y ejecución integradas.
- Task tool: spawnea sub-agentes con contexto fresco.
- MCP nativo para extender con servers externos.

OpenCode
open source · multi-provider

- Alternativa open source a Claude Code.
- Compatible con múltiples LLM providers.
- Misma filosofía de agente de terminal.
- Ideal para equipos con restricciones de vendor.

Skills Registry — Del Monolito a Módulos

Antes	Después
AGENTS.md monolítico	**Skills Registry + Router**
1000+ líneas leídas en CADA turno	Router detecta la tarea requerida.
El agente carga Angular skills aunque solo haga testing.	Carga SOLO las skills relevantes (~200 líneas).
Context window desperdiciada en instrucciones irrelevantes.	Skill de Angular solo si hay código Angular.
Compactación temprana → alucinaciones.	Context limpio → respuestas más precisas.

Engram — Resolviendo la Amnesia

Sistema de memoria persistente para agentes de IA

Go binary · SQLite + FTS5 · BM25 search · MCP Server · Agent-agnostic

What / Why / Where / Learned

El agente estructura lo que decide recordar en 4 dimensiones

Progressive Disclosure

El contexto se carga por relevancia, no todo de golpe

3 capas de resiliencia

Ante compactación: memoria externa, hints en contexto, topic keys

Progressive Disclosure

FORMATO DE MEMORIA:

what	why	where	learned
Implementé el Skills Registry en Claude Code con router basado en regex	El AGENTS.md crecía sin control. El contexto se saturaba en cada turno	Prowler frontend · src/agents/skills-router.md	Un router liviano cargando 200 líneas beats 1000 siempre

Human in the Loop (HITL)

Tony Stark tiene la visión. Jarvis ejecuta con precisión.

Anatomía gradual

El agente gana autonomía según demuestra confiabilidad en ejecuciones anteriores.

Puntos de aprobación

Antes de cada fase crítica el humano valida el output y decide si continuar.

No reemplaza — amplifica

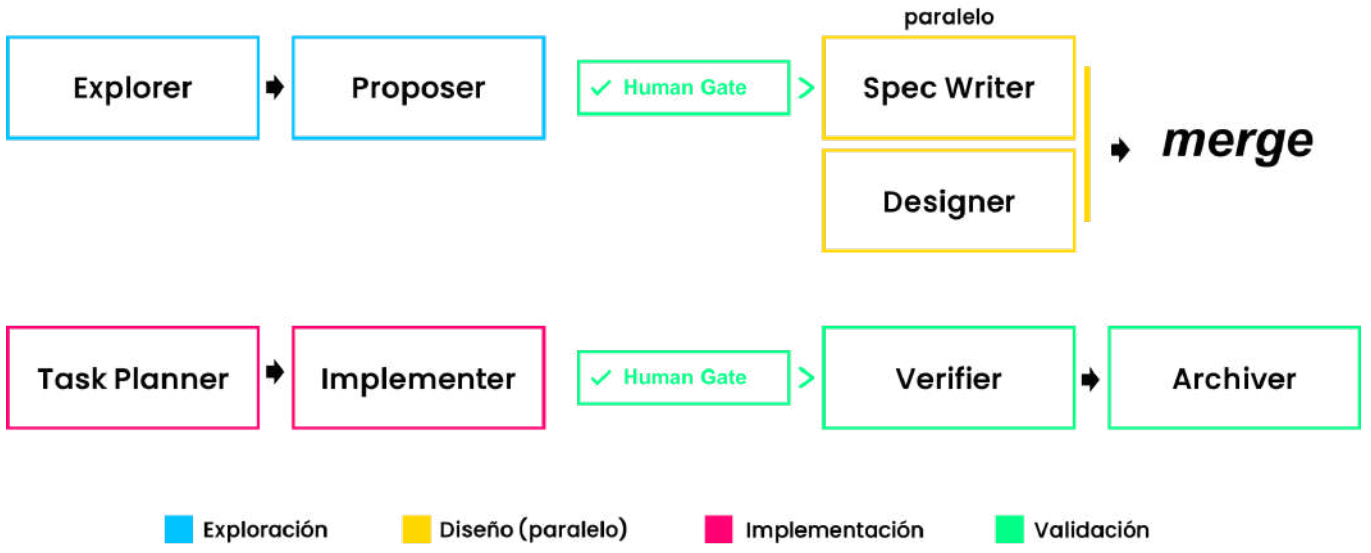
El objetivo es multiplicar la capacidad de decisión del humano, no eliminarlo.

Orquestadores Multi-Agente

El orquestador NUNCA hace trabajo real. Solo coordina.

NIVEL 1 BASIC SUBAGENTS	Task tool nativo de Claude Code. Un orquestador que spawna sub-agentes puntuales. Ideal para tareas independientes y paralelas.	**Claude Code nativo**
NIVEL 2 Agent Teams Lite	Pure Markdown. Zero dependencies. Orquestación declarativa vía archivos .md. Compatible con Claude Code / OpenCode / Cursor.	**Open Source** **Agent Teams Lite**
NIVEL 3 Full Agent Teams	DAG completo con fases secuenciales y paralelas. Contratos de resultado entre agentes. Persistencia con Engram. Human approval gates.	**Máxima potencia**

SDD Orchestrator — El Pipeline



Contratos de resultado
Cada sub-agente produce un output estructurado que el agente siguiente consume como input. Sin contratos no hay orquestación — solo caos.

Persistencia con Engram
Cada fase puede leer memoria de ejecuciones anteriores y escribir nuevos aprendizajes. El sistema mejora con cada run.

Fases paralelas
Spec Writer y Designer corren simultáneamente en sub-agentes independientes. El orquestador espera ambos outputs antes de continuar.

El Stack Cognitivo Completo

Todo Open Source · Todo Pure Markdown · Sin dependencias adicionales

Engram
Memoria persistente · SQLite · MCP · Agent-agnostic
https://github.com/Gentleman-Programming/engram

Agent Teams Lite
Orquestación declarativa · Markdown puro · Multi-provider
github.com/Gentleman-Programming/agent-teams-lite

Skills Registry
Router liviano · Carga bajo demanda · Sin context bloat
Incluido en gentle-ai

Human in the Loop
Approval gates · Autonomía gradual · El humano decide

Conclusiones

En 2026, la IA no ha eliminado al desarrollador; ha eliminado la mediocridad técnica. La tecnología se encarga de la mecánica, pero la responsabilidad de la visión, la ética y la calidad final recae en el humano. El éxito radica en dejar de ser un artesano del código para convertirse en un Arquitecto de Orquestación.

Los tres pilares del profesional de élite son:

- Dominio de Fundamentos Técnicos: Sin bases sólidas en arquitectura y patrones, es imposible auditar la producción de la "Legión".
- Maestría en Orquestación: Capacidad de configurar agent.md, MCPs, Skills y flujos multi-agente deterministas.
- Control Crítico (HITL): Mantener el juicio crítico sobre la estrategia, asegurando que el software no sólo funcione, sino que sea sostenible y escalable en producción.

BIG school

Súmate a nuestro directo del día 3

ACCEDE AQUÍ