DAVMAC SOLUTIONS

Guía práctica de desarrollo asistido por IA

VIBE CODE

Guía completa de desarrollo asistido por inteligencia artificial: de la idea al producto en producción

David Maceiras
DAVMAC Solutions

Índice

PARTE I

Fundamentos del Vibe Coding

Antes de escribir una sola línea de código asistido por inteligencia artificial, conviene entender qué es realmente el vibe coding, de dónde viene y por qué ha cambiado la forma en que se construye software. Esta primera parte sienta las bases: la definición del término, su evolución histórica, el ecosistema de herramientas disponible hoy y la mentalidad que separa a quien usa la IA como una calculadora de quien la usa como un colaborador real.

A lo largo de estos cuatro capítulos combinaremos teoría con ejemplos concretos, de forma que al terminar esta parte no solo entiendas los conceptos, sino que puedas reconocerlos la primera vez que te sientes a trabajar con una herramienta de IA.

Capítulo 1. Qué es el Vibe Coding

El término "vibe coding" fue popularizado en 2025 para describir una forma de programar en la que el desarrollador describe en lenguaje natural lo que quiere construir, y un modelo de inteligencia artificial genera, ejecuta y ajusta el código correspondiente, mientras la persona supervisa el resultado y corrige el rumbo mediante nuevas instrucciones. No se trata de escribir código a mano línea por línea, sino de mantener una conversación iterativa con un sistema capaz de escribir, probar y modificar software por sí mismo.

La palabra "vibe" (sensación, ambiente) no es casual: describe un modo de trabajo más fluido e intuitivo que la programación tradicional, donde el desarrollador se mantiene en un estado de flujo alto, avanzando por aproximaciones sucesivas en lugar de planificar cada detalle de antemano. Esto no significa improvisar sin criterio: significa mover el foco de atención de "cómo se escribe cada línea" a "qué debe hacer el sistema y cómo se verifica que lo hace bien".

Vibe coding no es lo mismo que autocompletado

Es habitual confundir el vibe coding con el autocompletado inteligente que ofrecen herramientas como GitHub Copilot desde 2021. La diferencia es de grado y de alcance. El autocompletado sugiere la siguiente línea o función mientras el desarrollador sigue escribiendo el resto a mano. El vibe coding, en cambio, delega unidades de trabajo completas —una función, un módulo, una funcionalidad entera— a un agente que puede leer el proyecto, escribir varios archivos, ejecutar comandos, ver los errores resultantes y corregirlos sin que el humano intervenga en cada paso.

Esta distinción importa porque cambia el rol del programador. En el autocompletado, el humano sigue siendo el autor principal del código y la IA es una ayuda puntual. En el vibe coding, el humano actúa más como director de producto y revisor de calidad: define el objetivo, evalúa el resultado, decide qué se acepta y qué se corrige, y la IA asume la autoría material de gran parte del código.

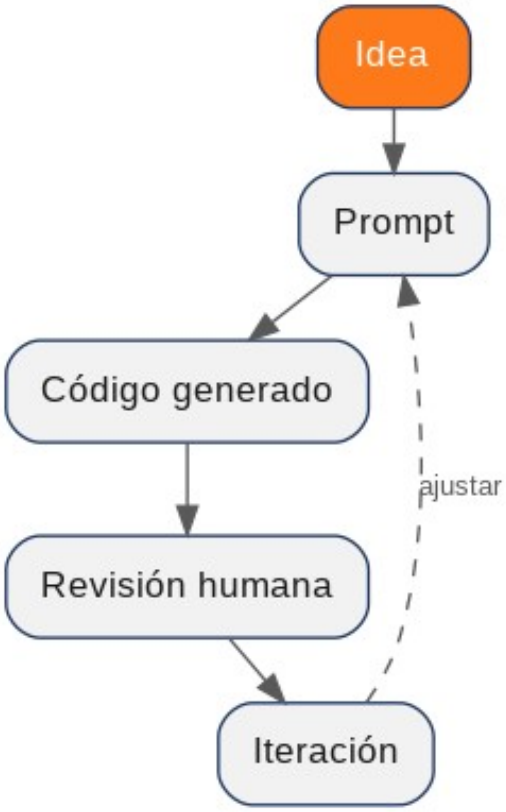

Figura 1.1 — El ciclo básico del vibe coding: de la idea a la iteración continua.

Los cuatro pilares del vibe coding

Para que este modo de trabajo funcione de forma fiable —y no se convierta en una sucesión de intentos aleatorios— conviene apoyarse en cuatro pilares:

- Comunicación precisa: cuanto más claro sea el objetivo que se transmite a la IA, menos iteraciones hacen falta para llegar a un resultado útil. Esto se desarrolla en profundidad en el Capítulo 5.
- Verificación constante: cada entrega de la IA se prueba, se ejecuta y se revisa antes de darla por buena. La confianza ciega en el código generado es la causa más común de fallos en producción.
- Contexto acumulado: los proyectos que funcionan bien con IA mantienen documentación y estructura que permiten a cualquier sesión nueva de trabajo "ponerse al día" rápidamente, sin depender de que el humano recuerde todo el histórico.
- Responsabilidad humana: quien aprueba un cambio es responsable de él, tanto si lo escribió una persona como si lo escribió un modelo de IA. El vibe coding no traslada la responsabilidad profesional, solo cambia quién teclea.

> **Idea clave**
> El vibe coding no elimina la necesidad de saber programar: la reduce en el día a día,

pero la hace imprescindible en los momentos de revisión, arquitectura y decisión. Cuanto mejor entiendas lo que la IA genera, mejor sabrás cuándo confiar en ello y cuándo no.

Quién practica vibe coding hoy

Aunque el término nació asociado a founders solitarios construyendo productos de fin de semana, en 2026 el vibe coding se practica en contextos muy distintos: equipos de producto que prototipan funcionalidades antes de comprometer recursos de ingeniería, agencias de desarrollo que reducen los tiempos de entrega a sus clientes, departamentos internos que automatizan procesos sin depender de un equipo técnico saturado, y desarrolladores experimentados que usan la IA para eliminar el trabajo repetitivo y concentrarse en las decisiones que de verdad requieren su criterio.

Lo que comparten todos estos casos es un cambio de cuello de botella: antes, la limitación para construir software era la disponibilidad de programadores; ahora, cada vez más, la limitación es la claridad con la que se define lo que hay que construir y la disciplina para revisarlo bien.

Un primer ejemplo de principio a fin

Para hacer tangible todo lo anterior, veamos un ejemplo mínimo. Supongamos que queremos una función que valide direcciones de email antes de guardarlas en una base de datos. En programación tradicional, escribiríamos la función nosotros mismos. En vibe coding, el proceso es distinto:

- El desarrollador describe el objetivo: "necesito una función en Python que valide direcciones de email, rechace formatos incorrectos y devuelva un mensaje de error claro en cada caso".
- La IA genera una primera versión de la función, junto con un pequeño conjunto de pruebas.
- El desarrollador ejecuta las pruebas y observa que un caso límite —una dirección con un punto justo antes de la arroba— no se maneja bien.
- El desarrollador comunica ese caso concreto a la IA, que ajusta la función y añade una prueba para ese caso específico.
- El desarrollador revisa el resultado final, lo entiende, y solo entonces lo integra en el proyecto.

```
def validar_email(direccion: str) -> tuple[bool, str]:
    """Valida una dirección de email básica.
    Devuelve (True, "") si es válida, o (False, motivo) si no lo es.
    """
    import re
    patron = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
    if not direccion or "@" not in direccion:
        return False, "Falta el símbolo @"
    if ".." in direccion or direccion.startswith(".") :
```

```
        return False, "Formato de puntos inválido"
    if not re.match(patron, direccion):
        return False, "Formato de email no reconocido"
    return True, ""
```

Este ejemplo, deliberadamente sencillo, contiene ya los cinco pasos que se repiten en proyectos mucho más grandes: definir, generar, probar, corregir y revisar. La única diferencia entre este caso y uno complejo es la cantidad de ciclos necesarios, no la naturaleza del proceso.