

VIBE CODING: IL FUTURO CONVERSAZIONALE DELLO SVILUPPO SOFTWARE -PRIMA PARTE-



A cura di Massimo Nannini ()*

Sommario

Introduzione: L'alba della programmazione intuitiva

Dalle parole al codice: Meccanismi e flussi di lavoro

Un ecosistema in rapida evoluzione:

Strumenti e piattaforme a confronto

Vibe Coding in azione: Esempi pratici e studi di caso

Analisi Critica: I pro, i contro e le implicazioni nascoste

Il Futuro dello Sviluppo Software: Proiezioni ed evoluzione del ruolo umano

Conclusioni e Raccomandazioni Strategiche

Il Vibe Coding trasforma la programmazione in un dialogo con l'intelligenza artificiale, sostituendo la scrittura manuale con la generazione di codice tramite LLM. Questo approccio accelera la prototipazione, amplia l'accesso anche ai non-programmatori e ridefinisce il ruolo dello sviluppatore, che diventa architetto e supervisore del processo. Opportunità e rischi coesistono: velocità e creatività da un lato, debito tecnico e sfide di manutenibilità dall'altro.

Introduzione: L'alba della programmazione intuitiva

Il panorama dello sviluppo software sta attraversando una trasformazione radicale, allontanandosi dal tradizionale processo di scrittura manuale del codice riga per riga per abbracciare un nuovo paradigma: l'interazione conversazionale con l'intelligenza artificiale. Al centro di questo cambiamento si trova il "Vibe Coding", una pratica emergente che sfrutta i modelli linguistici di grandi dimensioni (LLM) per generare codice funzionale a partire da prompt in linguaggio naturale. L'obiettivo principale è quello di accelerare il ciclo di sviluppo e rendere la creazione delle applicazioni più accessibile, in particolare per coloro che hanno un'esperienza di programmazione limitata.

Il termine "vibe coding" è stato coniato e reso popolare dal ricercatore AI Andrej Karpathy nel febbraio 2025, attraverso una serie di osservazioni pubblicate su X. La sua celebre frase, che invita a "dimenticare che il codice esista" e a "darsi completamente alle vibrazioni", incapsula la filosofia di un flusso di lavoro in cui il ruolo primario si sposta dalla meticolosa implementazione manuale quale codifica di un algoritmo in precedenza sviluppato, alla guida di un assistente AI. Sebbene Karpathy sia riconosciuto come il padre del neologismo, la pratica di utilizzare strumenti AI per la generazione di codice era già in fase di sperimentazione con l'uscita di modelli come ChatGPT e GitHub Copilot.

L'adozione del termine non è casuale. Prima della sua ridefinizione da parte di Karpathy, "vibe coding" era un termine informale, spesso usato in tono scherzoso all'interno delle comunità di sviluppatori, per descrivere un approccio di programmazione basato sull'intuizione e sullo "stato di flow", senza una pianificazione formale o documentazione rigorosa. Questa sovrapposizione concettuale evidenzia una profonda convergenza culturale nel settore tecnologico. Un'abitudine di sviluppo considerata disordinata e non standard nel passato, l'affidarsi al "sentimento" piuttosto che alla formalità, è stata riqualificata come un approccio legittimo e potenziato dall'IA. La capacità di un modello di tradurre un'intuizione vaga in codice funzionale trasforma quello che un tempo era un rischio (codice non pianificato) in un mezzo per ottenere velocità e innovazione.

Dalle parole al codice: Meccanismi e flussi di lavoro

Il Vibe Coding si basa su un ciclo di sviluppo iterativo e conversazionale, definito come un "loop stretto" tra lo sviluppatore e l'assistente AI. Questo processo si articola in quattro fasi principali:

- **Descrivere l'obiettivo:** Il punto di partenza è un prompt di alto livello, formulato in linguaggio naturale. Ad esempio, si può chiedere all'IA di "Creare una funzione Python che legge un file CSV".
- **L'IA genera il codice:** L'assistente interpreta la richiesta e produce un blocco di codice iniziale. L'utente può semplicemente inserire un commento nel proprio IDE per ottenere una funzione completa.
- **Eseguire e osservare:** Lo sviluppatore esegue il codice generato per verificarne il funzionamento e identificare eventuali bug o discrepanze rispetto all'intento originale.
- **Fornire feedback e raffinare:** Se il risultato non è soddisfacente, si forniscono nuove istruzioni, come "Aggiungi la gestione degli errori per quando il file non viene trovato". Questo ciclo di descrizione, generazione, test e affinamento si ripete finché il codice non è completo e soddisfa i requisiti.

In pratica, esistono due modi principali di applicare il Vibe Coding: il "Vibe Coding puro" e lo "sviluppo assistito da AI responsabile". Il primo è un approccio esploratorio, in cui l'utente si fida ciecamente dell'output dell'AI per progetti rapidi e "usa e getta", come i prototipi oppure valutazioni funzionali di alcune caratteristiche essenziali. Questo è ciò che Karpathy ha descritto come "dimenticare che il codice esista". Il secondo, invece, rappresenta l'applicazione professionale, dove l'AI agisce come un potente collaboratore o "pair programmer". In questo modello, lo sviluppatore mantiene il pieno controllo, esaminando, testando e comprendendo il codice generato.

La differenza fondamentale tra il Vibe Coding e il tradizionale Pair Programming risiede nella comprensione del codice. Mentre nel Pair Programming due esseri umani lavorano insieme sulla stessa codebase, con una piena e reciproca comprensione di ogni riga, il Vibe Coding implica che lo sviluppatore eviti di "micromanaggiare" il codice, accettando spesso l'output dell'IA senza una comprensione

completa del “come” e del “perché” il codice funzioni. Come ha osservato il programmatore Simon Willison, l'utilizzo dell'IA come un semplice “assistente di digitazione”, dove si esamina e si comprende ogni riga generata, non è Vibe Coding. Al contrario, il Vibe Coding implica un'accettazione più liberale del risultato, con un focus sull'esperienza iterativa piuttosto che sulla correttezza strutturale.

Nonostante le preoccupazioni sulla mancanza di comprensione, il Vibe Coding può anche fungere da potente strumento di apprendimento. Diversi ingegneri hanno notato che questa pratica è un modo efficace per i programmatori di acquisire familiarità con linguaggi e tecnologie che non conoscono. Invece di dover studiare una nuova sintassi da zero per un piccolo progetto, un professionista può generare rapidamente un codice funzionante e concentrarsi sulla logica del problema. Attraverso l'osservazione e il debugging del codice generato, il programmatore acquisisce rapidamente una comprensione del comportamento del sistema, accelerando la curva di apprendimento e rendendo fattibili “sidequests” che prima non lo sarebbero state.

Un ecosistema in rapida evoluzione: Strumenti e piattaforme a confronto

Il mercato del Vibe Coding è un ecosistema in rapida evoluzione, popolato da una varietà di strumenti e piattaforme che si rivolgono a diverse tipologie di utenti e casi d'uso. Si

possono categorizzare questi strumenti in tre gruppi principali.

- **Assistenti AI integrati negli IDE (IDE-First):** Questi strumenti sono progettati per integrarsi direttamente negli ambienti di sviluppo esistenti, supportando gli sviluppatori professionisti nel loro flusso di lavoro quotidiano. Ne sono esempi:
 - **Cursor:** Basato sul codice sorgente di Visual Studio Code, integra modelli come GPT-4 e Claude. Offre funzionalità avanzate di chat, refactoring e debugging per interi file.
 - **GitHub Copilot:** Uno dei pionieri del settore, lanciato nel 2021.9 Fornisce suggerimenti contestuali in tempo reale e può automatizzare compiti ripetitivi.
 - **GeminiCodeAssist:** Progettato per utenti intermedi e avanzati, offre assistenza in-editor, può generare test unitari e aiutare nel debugging direttamente all'interno dell'IDE.
 - **Aider:** Uno strumento basato su terminale, ideale per utenti avanzati che desiderano un'integrazione profonda con il proprio ambiente di sviluppo locale.
- **Piattaforme “End-to-End” (App Builder):** Queste soluzioni si rivolgono a un pubblico più ampio, dai non-programmatori ai designer e agli imprenditori che cercano di costruire e distribuire un'intera applicazione da un singolo prompt in un'unica interfaccia web.



- Lovable: Considerato uno dei più user-friendly per i non-coder 15, consente di costruire e distribuire un'intera app e di apportare modifiche mirate a elementi specifici dell'UI.
- Bolt.new: Ideale per chi ha un background di design, in quanto può importare progetti da Figma e convertirli in codice funzionale.
- Replit AI: Offre un ambiente di sviluppo completo che consente di creare, codificare, collaborare e distribuire app direttamente dal browser.
- Google AI Studio e Firebase Studio: Forniscono un percorso rapido e diretto per trasformare un'idea in un'applicazione web live, spesso con un solo prompt, e si concentrano su approcci no-code e low-code.
- Agenti AI Specializzati: Questi sono strumenti progettati per compiti specifici, spesso con un approccio "agente" o semi-autonomo.
 - Vercel v0: Si specializza nella generazione di interfacce utente (UI) di alta qualità, in particolare in React.
 - Sweep: Un agente che trasforma automaticamente le issue di GitHub in pull request, agendo come un "junior developer virtuale" per automatizzare bug fix e implementazioni.

L'analisi di questo ecosistema rivela che "Vibe Coding" non è un termine univoco, ma piuttosto un concetto che racchiude una vasta gamma di pratiche di sviluppo assistite dall'IA. I vari strumenti si rivolgono a un mercato segmentato per livello di competenza e per caso d'uso, dimostrando che non si tratta di un'unica soluzione, ma di un ventaglio di opzioni in cui la scelta dipende dagli obiettivi specifici, dalla complessità del progetto e dall'esperienza dell'utente. Questa segmentazione è un indicatore della maturità del mercato e della sua crescente adozione in diversi contesti professionali.



(*) Massimo Nannini
 IT engineer and business consultant,
info@gemaxconsulting.it

Strumento	Livello Utente	Approccio	Funzionalità Principali	Modalità
Cursor	Intermedio/Esperto	AI-assisted	Generazione, chat, refactoring, debugging	IDE (VS Code)
GitHub Copilot	Intermedio/Esperto	AI-assisted	Completamento codice, suggerimenti contestuali	IDE
Gemini Code Assist	Intermedio/Esperto	AI-assisted	Generazione, test, debugging, assistenza in-editor	IDE
Lovable	Principiante	No-code / Low-code	Generazione app end-to-end, modifica UI visuale	Web
Bolt.new	Principiante/Intermedio	No-code / Low-code	Figma-to-code, generazione UI/backend	Web
Replit AI	Principiante/Intermedio	No-code / Low-code	Generazione app end-to-end, deploy	Web
Google AI Studio	Principiante	No-code / Low-code	Generazione app single-prompt, deploy in un clic	Web
Vercel v0	Intermedio/Esperto	No-code / Low-code	Generazione UI in React	Web
Aider	Esperto	AI-assisted	Integrazione profonda con codebase locale e Git	Terminale