Code Buddy Narratif

Construire un agent IA de production

Patrice Huetz

patrice-huetz.fr

patrice-huetz.fr · contact@patrice-huetz.fr

CHAPITRE 1

Premier Contact

Six mois après la fin du livre sur les agents LLM. Bureau de Lina, un jeudi soir...

Lina avait passé deux heures à déboguer le même problème. Une régression subtile dans un module de paiement — le genre de bug qui se cache dans les interactions entre trois fichiers et une configuration d'environnement. Elle avait ouvert GitHub Copilot, lui avait décrit le problème. Il lui avait suggéré des modifications dans le mauvais fichier. Elle avait essayé ChatGPT en mode web. Utile pour expliquer le problème, inutile pour explorer sa propre codebase.

Son téléphone vibra. Marcus.

— "T'es encore au bureau ?"

— "Bug de merde. Copilot m'emmène en bateau."

— "Essaie Code Buddy. J'ai mergé la v0.1.19 ce matin."

Lina fronça les sourcils. Code Buddy. Elle avait vu le nom passer sur GitHub, l'avait ajouté à sa liste mentale de "trucs à tester un jour". Ce soir semblait être ce jour.

Elle clona le dépôt, suivit les instructions d'installation — cinq minutes, une variable d'environnement pour sa clé Anthropic — et lança le terminal.

```
$ code-buddy
```

Le CLI s'ouvrit. Interface propre, inspirée de Claude Code. Un prompt simple.

Elle tapa ce qu'elle aurait tapé à un collègue senior :

```
J'ai une régression dans le module de paiement. Les transactions
avec Stripe échouent de manière intermittente pour les montants > 100€ mais
seulement en production. Le bug n'apparaît pas dans les tests unitaires.
```

Ce qui se passa ensuite lui fit légèrement ouvrir la bouche.

L'agent ne répondit pas immédiatement par du texte. Il commença à agir.

```
🔍 Analyse de la codebase...
   → Lecture de src/payments/stripe.ts
   → Lecture de src/payments/validators.ts
   → Lecture de config/stripe.prod.json
   → Recherche pattern: "stripe" AND "amount"
   → 23 fichiers analysés, 4 pertinents identifiés

🔎 Recherche des logs de production...
   → Exécution: grep -r "stripe" logs/prod-2025-01.log | grep "error"
   → 12 occurrences trouvées

💡 Hypothèse : Problème de précision flottante
   → src/payments/stripe.ts ligne 47 : amount * 100 (conversion en centimes)
   → Pour les montants > 100€, la multiplication génère des nombres flottants
   → Stripe rejette les montants non entiers
```

Lina s'arrêta. Elle regarda la ligne 47 de son fichier. La multiplication. Elle n'y avait jamais pensé.

— "Oh bon sang," murmura-t-elle.

```
Voulez-vous que j'applique la correction ?
   → Math.round(amount * 100) au lieu de amount * 100
   → + ajout d'un test unitaire pour les montants avec décimales
   [O]ui  [N]on  [Voir le diff]
```

Elle tapa `O`.

Deux minutes plus tard — temps qu'elle passait habituellement à retrouver où était son café — le bug était corrigé, un test l'accompagnait, et l'agent lui proposait de vérifier si d'autres endroits dans le code effectuaient des conversions similaires.

Lina reposa ses doigts sur le bureau et contempla son écran. Ce n'était pas de la complétion de code. Ce n'était pas un chatbot. C'était quelque chose d'autre.

Elle appela Marcus.

— "Je veux comprendre comment ça marche. Vraiment. Pas juste l'utiliser."

> *Marcus sourit au téléphone. Il avait attendu cette conversation.*

Table des Matières

1.1 Positionnement : Code Buddy dans l'Écosystème

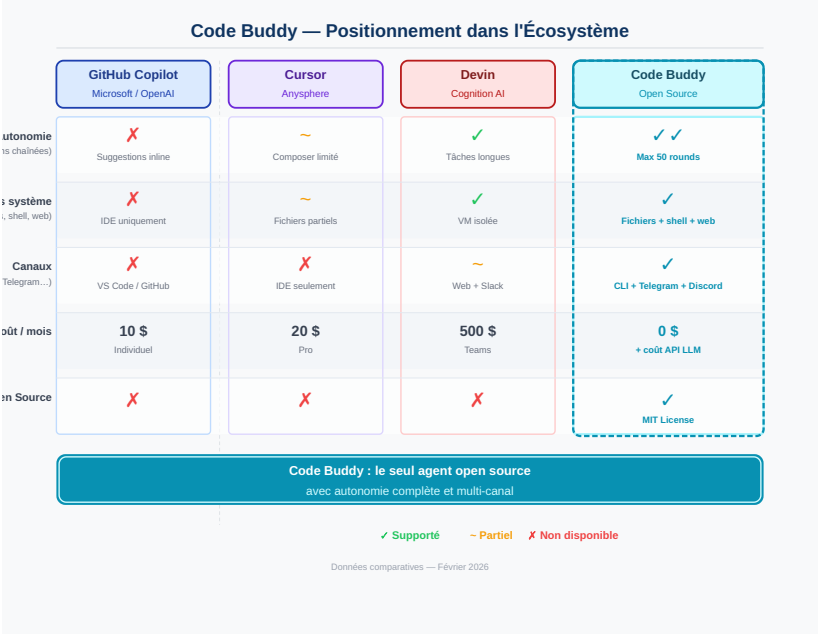
Le lendemain matin, dans le café habituel. Marcus avait apporté son ordinateur et un tableau comparatif qu'il avait visiblement préparé la veille.

— "Avant de plonger dans l'architecture," dit-il en poussant la feuille vers Lina, "il faut comprendre *pourquoi* Code Buddy existe. Il y a plein d'outils d'IA pour le code. Qu'est-ce qui justifie un de plus ?"

Lina parcourut le tableau.

Les Différentes Familles d'Outils IA pour le Code

Outil	Paradigme	Force	Limite
GitHub Copilot	Auto-complétion inline	Intégration IDE parfaite	Contexte limité au fichier ouvert
Cursor	IDE complet IA-native	UX très soignée	Propriétaire, pas d'API
ChatGPT / Claude	Chat généraliste	Raisonnement puissant	Ne voit pas ta codebase
Devin	Agent autonome cloud	Tâches très complexes	$500/mois, cloud seulement
Aider	CLI agent	Open source, rapide	Interface minimaliste
Code Buddy	Agent multi-modal	Open source + multi-canal + sécurité	v0.1.x, écosystème naissant



Positionnement de Code Buddy dans l'écosystème des outils IA
Code Buddy se distingue par son autonomie complète, son accès système total et son architecture open source multi-canal.

— "Vois-tu le pattern ?" demanda Marcus. "Chaque outil fait un choix fondamental. Copilot optimise pour la vitesse en contexte réduit. Devin optimise pour l'autonomie totale mais à un prix prohibitif. Code Buddy fait le pari d'une *surface de contact maximale* avec ton environnement réel."

— "C'est-à-dire ?"

— "Il peut t'envoyer un message sur Telegram depuis ton serveur de prod. Il peut surveiller un écran, prendre des screenshots, remplir des formulaires. Il tourne en daemon, exécute des tâches planifiées. Il peut gérer 10 utilisateurs en parallèle sur Discord avec des sessions isolées. C'est moins un assistant de code qu'un *développeur autonome* que tu héberges toi-même."

Tom, le DevOps de l'équipe qui passait par là, s'arrêta :

— "Vous parlez de Code Buddy ? Je l'ai regardé hier. Ce qui m'a convaincu, c'est le sandboxing Docker. Les autres outils CLI — ils exécutent n'importe quelle commande bash que le LLM génère. Code Buddy valide d'abord la syntaxe AST, demande confirmation pour les opérations dangereuses, et redirige les trucs vraiment risqués vers un container isolé."

Marcus hocha la tête.

— "Tom a mis le doigt sur l'enjeu central. Un agent qui peut modifier des fichiers et exécuter du code, c'est puissant. Mais c'est aussi terrifiant si le modèle hallucine un `rm -rf`. La sécurité n'est pas une réflexion d'après-coup dans Code Buddy — c'est une couche architecturale fondamentale."

1.2 Anatomie du Projet

Lina avait ouvert le dépôt dans son explorateur de fichiers. Elle essayait de comprendre la structure avant de plonger dans le code.

```
code-buddy/
├── src/
│   ├── agent/           # 🧠 Cerveau — CodeBuddyAgent, AgentExecutor
│   ├── tools/           # 🛠️  52+ outils (fichiers, shell, web, desktop...)
│   ├── providers/       # 🤖 Clients LLM (Anthropic, OpenAI, Gemini, Ollama...)
│   ├── security/        # 🔒 AST validation, confirmation gates, Docker sandbox
│   ├── rag/             # 📚 HNSW vector store, embeddings, compression
│   ├── channels/        # 📡 Telegram, Discord, Slack, WhatsApp, Matrix...
│   ├── memory/          # 💾 Mémoire long terme, checkpoints, apprentissage
│   ├── routing/         # 🔀 Model router, failover, parallel executor
│   ├── skills/          # 🎯 Système de skills en langage naturel
│   ├── cli/             # 💻 Interface terminal (Ink/React)
│   └── api/             # 🌐 REST/WebSocket server
├── db/                  # SQLite — 11 tables
├── skills/              # Fichiers SKILL.md des compétences bundlées
└── package.json         # TypeScript, Node.js ≥18, ~80 dépendances
```

— "C'est du TypeScript pur ?" demanda Lina.

— "TypeScript/Node.js. Ce qui est inhabituel pour un agent — la plupart sont en Python. Le choix de TypeScript permet d'utiliser Ink pour l'interface terminal, d'avoir du typage fort partout, et de partager du code entre le CLI et le serveur REST."

— "Et l'interface terminal est faite avec React ?"

— "Ink. C'est React mais pour le terminal. Les composants affichent du texte dans le terminal comme des composants React affichent du HTML dans le navigateur. C'est ce qui donne cet affichage propre avec les spinners, les boxes, les couleurs."

Stack Technique

Couche	Technologie	Pourquoi
Langage	TypeScript 5.x	Typage fort, écosystème Node.js
Runtime	Node.js ≥18	ESM natif, crypto natif, streams
Terminal UI	Ink (React)	Composants réutilisables pour CLI
Base de données	SQLite + better-sqlite3	Zéro config, persistence locale
Embeddings	HNSW via hnswlib-node	O(log n) search, 50× plus rapide que brute force
LLM Claude	@anthropic-ai/sdk	Provider principal
LLM OpenAI	openai	GPT-4, o1, o3
LLM Google	@google/generative-ai	Gemini Pro/Flash
Parsing AST	tree-sitter	Validation bash avant exécution
Recherche	ripgrep (via execa)	Grep ultra-rapide pour la codebase

1.3 Premiers Pas

Marcus ouvrit un terminal et montra à Lina l'installation complète.

Installation

```
# Via npm
npm install -g code-buddy

# Ou depuis les sources
git clone https://github.com/phuetz/code-buddy
cd code-buddy
npm install
npm run build
npm link
```

Configuration

Code Buddy utilise les variables d'environnement pour les clés API. Il détecte automatiquement le fournisseur disponible dans l'ordre de priorité :

```
# Dans ~/.bashrc ou ~/.zshrc
export ANTHROPIC_API_KEY="sk-ant-..."    # Claude (priorité 4)
export OPENAI_API_KEY="sk-..."           # GPT-4 (priorité 3)
export GOOGLE_API_KEY="..."              # Gemini (priorité 2)
export GROK_API_KEY="..."               # Grok (priorité 1)

# Pour les modèles locaux — aucune clé nécessaire
# Ollama : http://localhost:11434 (auto-détecté)
# LM Studio : http://localhost:1234 (auto-détecté)
```

— "L'ordre de priorité est Gemini → Grok → OpenAI → Anthropic ?" nota Lina. "C'est un peu surprenant."

— "C'est configurable. Par défaut, il favorise les modèles les plus économiques compatibles avec la tâche. Mais pour les tâches complexes, le router bascule automatiquement vers Claude ou GPT-4. On verra ça en détail au chapitre 4."

Lancement

```
# Mode interactif (CLI)
code-buddy

# Dans un répertoire projet — il analyse le contexte automatiquement
cd mon-projet && code-buddy

# Mode API server
code-buddy --server --port 3000

# Mode daemon (tourne en arrière-plan)
code-buddy --daemon start

# Avec un mode spécifique
code-buddy --mode architect
```

1.4 Les Quatre Modes Agentiques

— "Il y a quatre modes ?" demanda Lina. "Pourquoi ?"

Marcus expliqua :

— "Parce que toutes les tâches ne nécessitent pas le même niveau d'autonomie. Si tu poses une question sur une API, tu n'as pas besoin que l'agent modifie des fichiers. Si tu lui demandes de refactoriser un module entier, tu veux qu'il planifie avant d'agir. Les modes permettent de contrôler le périmètre d'action."

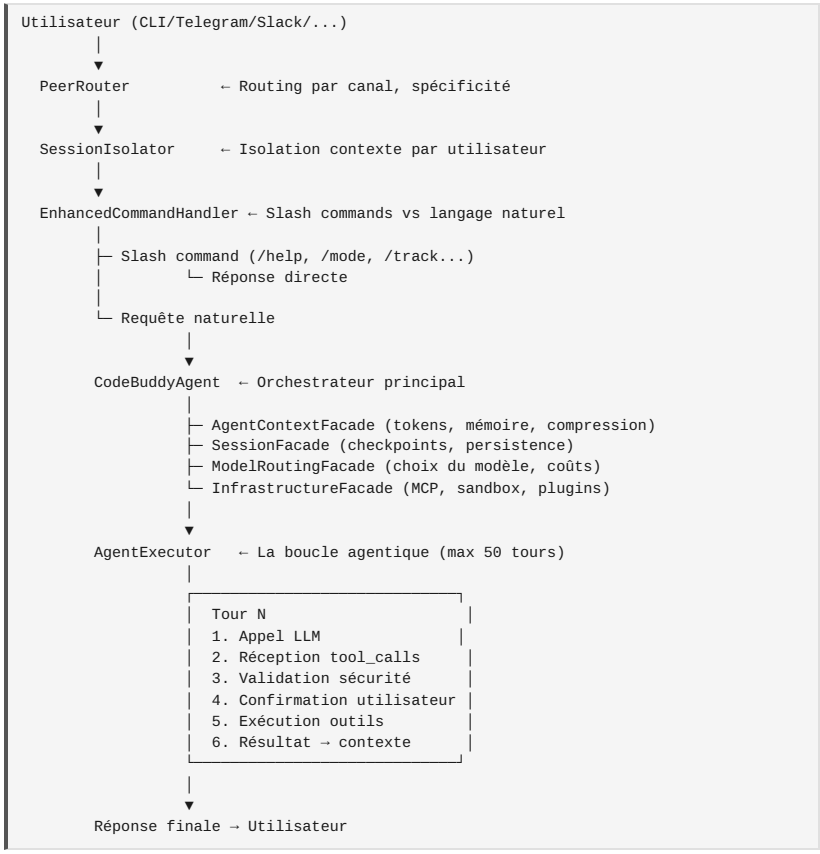
Mode	Commande	Capacités	Quand l'utiliser
ask	`/mode ask`	Lecture seule, réponses, explications	Questions, exploration, debug verbal
code	`/mode code`	Lecture + écriture fichiers + shell	Corrections, features, refactoring
plan	`/mode plan`	Lecture + génération de plans uniquement	Architecture, avant une grosse tâche
architect	`/mode architect`	Tout + supervision multi-agents	Projets complexes, revue d'ensemble

— "En mode `ask`, il ne peut pas toucher à tes fichiers même s'il le voulait," précisa Tom depuis sa table. "C'est une contrainte au niveau de l'outil, pas juste une consigne dans le prompt. Les outils d'écriture sont désactivés. C'est important — un LLM peut ignorer une instruction dans son prompt. Il ne peut pas ignorer l'absence d'un outil."

Cette distinction — contrainte par le code vs contrainte par le prompt — allait devenir un thème central dans les discussions qui suivraient.

1.5 Une Requête de A à Z

Avant de plonger dans les détails d'implémentation, Marcus voulait que Lina visualise le flux complet d'une requête. Il prit une serviette en papier — leur support de prédilection pour les schémas — et commença à dessiner.



— "50 tours par défaut," dit Lina. "Et en YOLO mode ?"

— "400."

Silence.

— "C'est quoi YOLO mode exactement ?"

Marcus sourit.

— "Le mode où tu fais confiance à l'agent. Plus de confirmations utilisateur, budget de tours maximal. Utile pour les tâches très longues que tu veux lancer la nuit. Dangereux si tu ne comprends pas ce que fait l'agent. Le nom n'est pas anodin."

1.6 Ce Qui le Différencie : Vue Chiffrée

Lina avait sorti son carnet. Elle aimait les chiffres concrets.

Caractéristique	Valeur	Contexte
Outils intégrés	52+	Copilot : 0, Aider : ~8
Fournisseurs LLM	6	Claude, GPT, Gemini, Grok, Ollama, LM Studio
Canaux de communication	10+	CLI, REST, WS, Telegram, Discord, Slack, WhatsApp, Signal, Matrix, Teams
Tables SQLite	11	Conversations, embeddings, stats, apprentissage
Réduction de coûts	30-70%	Via routing FrugalGPT + caching sémantique
Hit rate cache	60-70%	LSH semantic cache
Tours max standard	50	Configurable
Tours max YOLO	400	Pour tâches longues autonomes
Réduction tokens	7%	Via compression JetBrains 2024
Vitesse HNSW	50×	Vs brute force à 100K vecteurs

1.7 Points Clés

Lina referma son carnet. Il était 23h. Elle avait passé la soirée à explorer le code source avec Marcus, à tester des commandes, à poser des questions. Elle avait une dernière question :

— "Si je devais expliquer Code Buddy en une phrase à quelqu'un qui ne connaît rien aux agents IA ?"

Marcus réfléchit une seconde.

— "C'est un développeur senior que tu héberges sur ton serveur, qui connaît ta codebase mieux que toi, qui peut te joindre sur Telegram à 3h du matin si un déploiement plante, et qui te demande la permission avant de faire quoi que ce soit d'irréversible."

Lina hocha la tête lentement.

— "Et demain, on regarde comment les six couches s'assemblent ?"

— "Demain, on dissèque le moteur."

Ce qu'il faut retenir

Concept	Essentiel
Paradigme	Agent autonome multi-canal, pas complétion de code
Stack	TypeScript/Node.js, Ink/React terminal, SQLite
Différenciateur clé	Surface de contact maximale avec l'environnement réel
Sécurité	Contrainte par le code (désactivation d'outils), pas par le prompt
4 modes	ask (lecture) → code → plan → architect
Flux	Canal → PeerRouter → SessionIsolator → CodeBuddyAgent → AgentExecutor

Exercices

1.1 — Installez Code Buddy et lancez-le dans un projet existant. Demandez-lui de vous expliquer la structure du projet en mode `ask`. Observez quels fichiers il consulte.

1.2 — Comparez le comportement en mode `ask` vs `code` face à la même demande : *"Améliore la gestion d'erreurs dans le module principal."* Notez les différences.

1.3 — Examinez le fichier `src/agent/CodeBuddyAgent.ts`. Identifiez les quatre facades et cherchez comment elles sont initialisées (lazy loading).

Chapitre suivant : 🏗️ Les Six Couches