TANSA ÉDITIONS

AGENTIC CODING

AVEC CODEX

Construire des logiciels avec des agents IA sans perdre le contrôle

VICTOR PEREZ

AGENTIC CODING AVEC CODEX

Construire des logiciels avec des agents IA sans perdre le contrôle

EXTRAIT GRATUIT

Victor PEREZ

Édition 2026
Tansa Éditions

Sommaire

Introduction — Le jour où le code a commencé à répondre

Pendant longtemps, programmer signifiait apprendre une langue destinée aux machines. Il fallait connaître les mots exacts, respecter la ponctuation et ne surtout pas oublier un caractère. Une virgule égarée pouvait gâcher une soirée entière.

Puis les outils ont appris à comprendre le langage naturel.

Aujourd'hui, une personne peut écrire :

> Je veux que mon application traite les nouvelles données dès leur arrivée, sans interrompre les autres tâches.

Un agent de programmation peut examiner le projet, repérer les fichiers concernés, proposer un plan, modifier le code et exécuter des tests.

Cette évolution est spectaculaire, mais elle crée une illusion dangereuse : puisque l'agent sait produire du code, il suffirait de lui dire vaguement ce que l'on veut et d'attendre que tout fonctionne.

Ce serait comme engager un excellent constructeur, lui tendre les clés d'un terrain et lui dire :

> Fais-moi quelque chose de sympa. Je reviens vendredi.

Vendredi, il y aura peut-être une maison. Il y aura peut-être un parking. Il y aura peut-être une maison avec le parking dans le salon. Le problème ne vient pas forcément du constructeur. La mission était floue.

L'Agentic Coding est une manière de construire des logiciels en collaborant avec des agents capables d'agir.

L’agent ne se contente pas de suggérer une ligne de code : il peut explorer, planifier, modifier, tester et rendre compte.

Le rôle humain ne disparaît donc pas. Il se déplace.

L’humain doit :

- exprimer le résultat attendu ;
- fournir le contexte utile ;
- fixer les limites ;
- choisir le niveau d’autonomie ;
- contrôler les preuves ;
- prendre la décision finale.

Ce livre raconte comment mettre cette organisation en place avec ChatGPT et Codex. Il montre aussi comment nous l’avons appliquée pour construire Tansa, une application consacrée à l’analyse de données Bitcoin.

Le message central tient en une phrase :

> On peut déléguer l’écriture du code. On ne délègue pas la responsabilité du système.

Avant de commencer — Choisir son atelier Codex

Codex est l'agent de programmation d'OpenAI. Il peut lire un projet, rechercher un comportement, proposer un plan, modifier des fichiers, exécuter des commandes et vérifier son travail.

On peut cependant le rencontrer dans plusieurs ateliers. Le marteau reste un marteau, mais l'établi change.

Les principales façons de travailler

Surface	À quoi elle sert	Pour qui ?
Application ChatGPT ou Codex	Discuter, planifier, suivre les modifications et travailler avec des outils dans une interface visuelle	Débutants, responsables de projet et utilisateurs qui préfèrent éviter le terminal
Extension dans l'éditeur	Travailler avec Codex à côté des fichiers ouverts dans l'environnement de développement	Personnes qui utilisent déjà un éditeur de code
Codex CLI	Piloter Codex depuis un terminal dans un dépôt local	Utilisateurs à l'aise avec les commandes et Git
Travail cloud	Confier une tâche dans un environnement isolé et récupérer le résultat	Missions longues, parallèles ou détachées de la machine locale

Les fonctions disponibles peuvent dépendre de l'interface, du compte et de la configuration. Vous n'avez pas besoin

de maîtriser les quatre possibilités. Choisissez celle dans laquelle vous comprenez le mieux :

- quel projet est ouvert ;
- ce que Codex peut lire ;
- ce qu'il peut modifier ;
- les commandes qu'il exécute ;
- l'endroit où apparaissent le diff et les résultats.

Si vous débutez, une interface visuelle est souvent le chemin le plus doux. Si votre projet vit déjà dans un terminal et que `git status` ne vous donne pas de boutons, la CLI peut être naturelle.

Le premier contact

N'ouvrez pas votre projet le plus précieux en commençant par :

> Modernise tout et surprends-moi.

Choisissez une mission sans conséquence importante :

> Analyse ce projet sans modifier de fichier. Explique son objectif, ses principaux dossiers, la manière de le lancer et les tests disponibles. Utilise un langage simple et signale ce que tu n'as pas pu déterminer.

Cette première mission permet de vérifier quatre choses :

1. Codex travaille dans le bon dossier ;
2. il comprend l'organisation générale ;
3. il trouve les règles et les tests ;
4. son compte rendu est compréhensible.

Vous découvrez également sa manière de travailler sans lui confier immédiatement les clés de la fusée.

Instruction et protection technique

Écrire « ne modifie aucun fichier » est une instruction donnée à l'agent.

Choisir un environnement en lecture seule est une protection technique.

Les deux sont utiles, mais elles ne jouent pas le même rôle. L'instruction exprime votre intention. La protection limite concrètement les actions possibles. Pour une première exploration, utilisez la lecture seule lorsqu'elle est disponible.

Pour une modification locale, vous pourrez ensuite autoriser l'écriture dans le projet tout en conservant des limites :

- pas d'accès réseau inutile ;
- pas de commit ;
- pas de push ;
- pas de déploiement ;
- confirmation avant une suppression ou une action difficile à annuler.

Votre premier cycle de travail

Un cycle très simple suffit :

```
1. Décrire le résultat attendu.
2. Laisser Codex inspecter le projet.
3. Lire le plan ou l'explication.
4. Autoriser une modification limitée.
5. Examiner le diff et les tests.
6. Accepter, corriger ou abandonner.
```

Abandonner une modification n'est pas un échec. C'est précisément l'un des avantages d'un travail local et contrôlé : on peut refuser une direction avant qu'elle ne devienne le problème préféré des utilisateurs.

La documentation officielle des bonnes pratiques Codex présente le même principe général : fournir le bon contexte, planifier les tâches difficiles, rendre les règles durables et vérifier le résultat.

1. Un agent de programmation n’est pas un clavier magique

Un assistant classique répond à une question. Un agent, lui, peut poursuivre un objectif en utilisant des outils.

Si vous demandez à un assistant :

> À quoi sert un test logiciel ?

Il vous répondra.

Si vous demandez à Codex :

> Ajoute un test qui prouve que le paiement refusé ne crée aucune commande.

Il peut rechercher les fichiers concernés, comprendre l’organisation des tests, écrire le scénario, l’exécuter et corriger son travail si le test échoue.

Cette capacité d’action change la relation. Codex ressemble moins à un dictionnaire et davantage à un collaborateur qui dispose d’un terminal, d’un éditeur et d’un plan de travail.

Trois niveaux d’aide

On peut distinguer trois niveaux :

Le conseil. L’IA explique ce qu’il faudrait faire.

La production. L’IA écrit un fragment de code que l’humain copie.

L’action agentique. L’IA travaille directement dans le projet, utilise les outils disponibles, vérifie le résultat et poursuit la mission jusqu’à un état défini.

L’Agentic Coding commence véritablement au troisième niveau.

Cela ne rend pas les deux premiers inutiles. Demander une explication avant une modification reste souvent une excellente idée. On évite ainsi de découvrir après coup qu’une « petite optimisation » a déplacé la cuisine sur le toit.

Le terme Agentic Coding ne désigne pas ici une norme officielle avec un formulaire en trois exemplaires. Dans ce livre, il nomme une méthode de création dans laquelle un agent peut agir dans le projet, tandis que l’humain définit le but, les limites et les preuves.

L’autonomie n’est pas l’indépendance

Un agent autonome peut choisir plusieurs étapes pour atteindre un but. Il n’est pas pour autant indépendant de toute règle.

Une voiture peut maintenir sa vitesse automatiquement. Le conducteur choisit toujours la destination, la route et le moment où il faut s’arrêter.

Avec Codex, l’autonomie doit être entourée de quatre éléments :

- un objectif ;
- un périmètre ;
- des contraintes ;
- une définition du travail terminé.

OpenAI recommande justement de fournir à Codex le but, le contexte, les contraintes et les conditions indiquant que la mission est terminée. Cette structure réduit les suppositions et facilite la revue.

La première erreur : confondre résultat visible et résultat correct

Imaginons une boutique en ligne. Vous demandez à un agent d'ajouter un bouton rouge « Rembourser ». Le bouton apparaît. Mission accomplie ?

Pas forcément.

Il faut encore savoir :

- qui a le droit de cliquer ;
- si le remboursement est enregistré ;
- si le prestataire de paiement est appelé ;
- si un double clic rembourse deux fois ;
- si le client reçoit un message ;
- si l'action peut être auditée.

Le bouton est visible. Le système, lui, peut être faux.

L'Agentic Coding ne consiste donc pas à obtenir rapidement quelque chose qui ressemble au résultat. Il consiste à organiser le travail pour obtenir un résultat que l'on peut raisonnablement accepter.

Fiche pratique — Reconnaître une mission agentique

Une mission est adaptée à Codex lorsqu'elle possède :

- un résultat observable ;
- un projet auquel l'agent peut accéder ;
- des limites explicites ;
- une manière de vérifier le résultat.

Exemple faible :

> Améliore mon application.

Exemple exploitable :

> Réduis le temps d’affichage de la page des commandes sans modifier son contenu. Commence par mesurer les requêtes actuelles. La mission est terminée lorsque le nombre de requêtes est réduit, que les tests ciblés passent et que le diff ne contient aucune modification sans rapport.

3. Une équipe à trois : vous, ChatGPT et Codex

Dans la méthode présentée ici, trois rôles collaborent.

Cette séparation est une méthode de travail choisie pour ce livre. Elle ne décrit pas un mécanisme secret d'OpenAI et ne prétend pas révéler les instructions internes de Codex. Selon l'interface utilisée, ChatGPT et Codex peuvent apparaître dans un même environnement ou dans des outils distincts. Ce qui compte est de séparer trois responsabilités : comprendre le besoin, préparer la mission et agir dans le projet.

Vous connaissez le besoin

Vous pouvez ne pas connaître le nom du fichier à modifier. En revanche, vous savez souvent décrire ce que vous observez :

> Layer1 traite maintenant un bloc en quelques secondes, mais il s'arrête encore comme s'il était en retard.

Cette phrase contient une information précieuse : le comportement historique n'est peut-être plus adapté aux performances actuelles.

ChatGPT transforme l'intention en mission

ChatGPT sert de partenaire de réflexion. Il aide à :

- clarifier le besoin ;
- repérer les questions cachées ;
- distinguer un symptôme d'une cause ;
- traduire l'intention en objectif vérifiable ;

- préparer le prompt destiné à Codex ;
- interpréter le compte rendu technique.

Il joue ici un rôle proche de celui d'un architecte fonctionnel ou d'un responsable de mission.

Cette organisation est notre méthode de travail. OpenAI recommande la structure des bons prompts et l'utilisation d'un contexte adapté ; la séparation précise entre ChatGPT architecte et Codex exécutant est une manière pratique d'appliquer ces principes.

Codex travaille dans le projet

Codex peut ensuite :

- inspecter le dépôt ;
- suivre le chemin d'exécution ;
- consulter les règles du projet ;
- proposer un plan ;
- modifier les fichiers ;
- lancer les tests ;
- contrôler le diff ;
- produire un compte rendu.

Le cycle complet

Le fonctionnement ressemble à ceci :

1. Vous décrivez ce que vous voulez.
2. ChatGPT clarifie le problème.
3. ChatGPT prépare une mission structurée.
4. Codex analyse le projet.
5. Codex propose ou exécute la modification autorisée.

6 Codex fournit des résultats et des preuves.

7 ChatGPT traduit ces résultats et identifie les points à vérifier.

8 Vous décidez de poursuivre, corriger, committer ou abandonner.

On peut représenter ce cycle sans diagramme d'ingénieur nucléaire :

```
Besoin humain
     ↓
Clarification avec ChatGPT
     ↓
Mission vérifiable pour Codex
     ↓
Analyse → modification → tests
     ↓
Diff, preuves et limites
     ↓
Décision humaine
```

Le mouvement n'est pas toujours parfaitement linéaire. Une preuve insuffisante peut renvoyer vers une nouvelle mission. Une ambiguïté découverte par Codex peut revenir vers ChatGPT ou directement vers vous. Le cycle est une boucle de décision, pas un tapis roulant où l'on dépose une idée à gauche pour récupérer une application parfaite à droite.

Une comparaison avec un restaurant

Vous êtes le client qui sait quel repas il veut.

ChatGPT est le maître d'hôtel qui transforme « quelque chose de léger mais pas triste » en commande compréhensible.

Codex est la brigade qui connaît la cuisine, les outils et les recettes.

Git est la caméra de sécurité qui permet de savoir exactement qui a mis de l'ananas dans la carbonara.

Les tests sont la dégustation avant de servir.

Et la production est le client final. Évitons de lui demander de goûter en premier.

Fiche pratique — Ne pas mélanger les responsabilités

Question	Responsable principal
Quel problème mérite d'être résolu ?	Vous
Comment rendre la mission précise ?	Vous et ChatGPT
Où se trouve le comportement dans le code ?	Codex
Quelle modification semble adaptée ?	Codex propose, vous décidez
Les preuves sont-elles suffisantes ?	Vous, aidé par ChatGPT et Codex
Peut-on publier ?	Vous

6. Utiliser ChatGPT pour préparer une mission Codex

C’est ici que notre méthode prend toute sa valeur.

Vous n’êtes pas obligé de savoir produire immédiatement un prompt technique. Vous pouvez commencer avec vos mots :

> Même si Layer1 est rapide, il est obligé de s’arrêter souvent, non ?

Cette phrase est une question, pas encore une mission. ChatGPT doit effectuer plusieurs transformations.

Nous décrivons ici une méthode visible : vous donnez une demande à ChatGPT, vous examinez sa reformulation, puis vous transmettez la mission obtenue à Codex ou vous la faites exécuter dans l’environnement disponible. Nous ne décrivons pas les prompts internes ou cachés utilisés par OpenAI.

Étape 1 — Séparer le fait de l’hypothèse

Le fait peut être :

> Layer1 traite actuellement un bloc en dix à quinze secondes.

L’hypothèse peut être :

> Il perd du temps parce que le scheduler l’arrête entre les blocs.

La distinction est capitale. Si l’hypothèse est présentée comme une certitude, Codex cherchera peut-être uniquement des preuves qui la confirment.

Le prompt doit demander :

> Vérifie si les interruptions observées proviennent des règles du scheduler ou d'une autre cause.

Étape 2 — Retrouver l'intention

Le véritable besoin n'est pas « supprimer une condition dans un fichier ». Le besoin est :

> Lorsque Layer1 a du travail, il doit le poursuivre. Lorsqu'il n'en a plus, les autres modules peuvent utiliser les ressources.

Cette formulation exprime un comportement et non une implémentation.

Étape 3 — Identifier les inconnues

ChatGPT peut faire apparaître les questions que le premier message ne contenait pas :

- Qui démarre Layer1 ?
- Qui l'arrête ?
- Comment un nouveau bloc réveille-t-il le système ?
- Quelle priorité doit reprendre lorsque Layer1 est à jour ?
- Existe-t-il des protections contre deux exécutions simultanées ?

Toutes ces questions ne doivent pas forcément être posées à l'utilisateur. Certaines peuvent devenir des points d'inspection pour Codex.

Étape 4 — Définir l'autorisation

Il existe une différence majeure entre :

> Analyse la cause et propose une correction.

et :

> Analyse, implémente la correction, exécute les tests et prépare le commit.

Le premier prompt autorise une investigation. Le second autorise des modifications. Une mission doit préciser jusqu'où Codex peut aller.

Étape 5 — Construire les preuves attendues

ChatGPT ajoute ensuite les éléments qui permettront de juger le résultat :

- fichiers réellement touchés ;
- tests ciblés ;
- mesure avant/après si la mission concerne les performances ;
- risques ou limites ;
- état du dépôt ;
- absence de commit si une validation humaine est prévue.

Exemple complet

Demande initiale :

> Je veux que Layer1 construise les blocs au fur et à mesure qu'ils arrivent.

Mission construite :

```
Mission
Faire en sorte que Layer1 traite continuellement tout retard
  supérieur à zéro
et rende la priorité à Cluster uniquement lorsqu'il est à jour.

Contexte
Layer1 traite désormais un bloc en dix à quinze secondes, alors
  qu'un nouveau
bloc Bitcoin arrive en moyenne beaucoup moins fréquemment. Le
  mécanisme
historique de pause et de reprise semble créer des arrêts inutiles.
```

```
Analyse préalable
Retrace le chemin réel depuis l'arrivée d'un bloc jusqu'au réveil du
   scheduler.
Identifie les seuils d'arrêt et de reprise ainsi que les protections
   contre les
exécutions concurrentes. Ne suppose pas que l'hystérésis est
  l'unique cause.

Contraintes
- conserver l'ordre contractuel du pipeline ;
- ne pas modifier les autres politiques de priorité ;
- éviter toute refactorisation sans rapport ;
- ne créer aucun commit avant validation.

Terminé lorsque
- le comportement est couvert par des tests ciblés ;
- Layer1 reste actif tant que son retard est supérieur à zéro ;
- Cluster peut reprendre lorsque ce retard atteint zéro ;
- l'arrivée d'un nouveau bloc réveille le scheduler ;
- le diff ne contient que les changements nécessaires.

Compte rendu
Résume la cause, les fichiers modifiés, les tests exécutés, les
  résultats,
les limites restantes et l'état Git.
```

Lire l'anatomie du prompt

Le prompt final n'est pas meilleur parce qu'il est long. Il est meilleur parce que chaque bloc possède une fonction.

Bloc	Question à laquelle il répond	Risque évité
Mission	Quel résultat voulons-nous ?	Partir dans une autre direction
Contexte	Quels faits changent la décision ?	Deviner à partir d'informations incomplètes
Analyse préalable	Que faut-il vérifier avant d'écrire ?	Transformer une hypothèse en correction
Contraintes	Que devons-nous protéger ?	Gagner en vitesse en cassant une garantie
Terminé lorsque	Quelles preuves accepterons-nous ?	S'arrêter dès que le résultat « a l'air bon »

Bloc	Question à laquelle il répond	Risque évité
Compte rendu	Que devra comprendre l'humain ?	Recevoir un résumé flatteur mais inutilisable

Deuxième exemple — Une boutique sans vocabulaire technique

Demande initiale :

> Des fois le stock devient négatif et je ne veux plus que ça arrive.

ChatGPT peut commencer par distinguer :

- le fait : une quantité négative a été observée ;
- l'hypothèse : deux ventes simultanées ou une absence de contrôle pourraient l'expliquer ;
- l'intention : empêcher une sortie supérieure au stock disponible ;
- la question ouverte : faut-il refuser la vente ou seulement avertir ?

Si la règle produit est de refuser la sortie, la mission peut devenir :

```
Mission
Empêcher toute sortie de stock qui rendrait la quantité disponible
  négative.

Contexte
Une quantité négative a été observée dans l'application. La cause
  n'est pas
encore confirmée. Plusieurs ventes peuvent être enregistrées à peu
  d'intervalle.

Commence par
Retracer le chemin d'une sortie de stock depuis l'interface jusqu'à
l'enregistrement. Vérifie où la quantité disponible est contrôlée et
   si deux
opérations peuvent se chevaucher.
```

```
Contraintes
- conserver l'historique des mouvements ;
- ne pas corriger les données existantes sans mission séparée ;
- ne pas modifier les autres règles de vente ;
- ne créer aucun commit.

Terminé lorsque
- une sortie supérieure au stock est refusée avec un message
  compréhensible ;
- deux sorties simultanées ne peuvent pas contourner la règle ;
- les tests ciblés prouvent le refus et le cas normal ;
- le diff reste limité au comportement demandé.

Compte rendu
Explique la cause, la protection choisie, les fichiers modifiés, les
   tests
exécutés et ce qui reste non vérifié.
```

Le lecteur n'a pas eu besoin de connaître le nom d'une transaction de base de données ou d'un verrou. Il a dû connaître la règle : le stock ne devient pas négatif. Codex peut découvrir la traduction technique dans le projet.

Ce que ChatGPT doit parfois vous demander

ChatGPT ne doit pas transformer chaque phrase en prompt sans vous déranger. Certaines réponses changent le produit :

- faut-il refuser ou avertir ?
- qui possède le droit d'effectuer l'action ?
- peut-on modifier les données anciennes ?
- quelle perte de précision est acceptable ?
- une interruption de service est-elle autorisée ?

Lorsqu'une réponse modifie une règle métier, une conséquence financière ou une action difficile à annuler, la question doit revenir vers l'humain. Une bonne clarification ne consiste pas à tout demander. Elle consiste à demander ce que l'agent ne doit pas décider seul.

Montrer les coulisses dans le livre

Pour chaque grande mission Tansa, nous présenterons :

1. la phrase originale de l'utilisateur ;
2. ce que ChatGPT en comprend ;
3. les ambiguïtés détectées ;
4. le prompt final ;
5. la réponse de Codex ;
6. l'interprétation de ChatGPT ;
7. la décision humaine.

Le lecteur ne verra pas seulement « le bon prompt ». Il verra comment celui-ci a été construit.