ORCHESTRER LE CODE

Architecture, Git, tests, agents et mise en production expliqués à ceux qui décident

La promesse
Comprendre assez profondément le logiciel pour donner une direction claire à un agent de code, vérifier son travail et publier sans jouer à la roulette russe.

MANUEL DE CHEVET • ÉDITION 2026

Une méthode de vibe coding intelligent

Avant-propos

Ce livre n’essaie pas de faire de toi un développeur en accéléré. Il vise une compétence différente : devenir le maître d’ouvrage lucide d’un système logiciel. Tu dois savoir poser le problème, découper le travail, reconnaître une décision structurante, demander une preuve et refuser une mise en production mal préparée.

Un agent sait produire beaucoup de code. Il ne connaît pourtant ni ton entreprise, ni le coût futur d’une dépendance, ni la gravité réelle d’une perte de données. Sa vitesse amplifie la qualité de ta méthode - ou l’absence de méthode. Le vrai levier n’est donc pas le prompt magique. C’est le harnais : contexte, règles, étapes, tests, revues, traces et limites.

> Idée centrale
> Le code est un matériau. L’architecture organise ce matériau. Git garde son histoire. Les tests apportent des preuves. Le déploiement transforme une version vérifiée en service réel.

Sommaire

CHAPITRE 1

Voir le logiciel comme un système

Avant d’écrire une ligne, il faut comprendre ce qui circule, ce qui décide, ce qui persiste et ce qui peut casser.

Une application est une organisation

Imagine un atelier. L’interface est le comptoir. La logique métier est le savoir-faire. La base de données est l’archive. L’API est le bordereau qui fait circuler les demandes. Le serveur est le bâtiment équipé. Le réseau est la route. Une application fiable attribue clairement chaque responsabilité.

Les quatre mouvements fondamentaux

1. Recevoir
   Une intention entre : clic, formulaire, fichier, appel API.
2. Valider
   Le système vérifie format, droit, cohérence et règles métier.
3. Transformer
   Il calcule, génère, classe ou déclenche une action.
4. Persister ou répondre
   Il enregistre un état durable et retourne un résultat observable.

État, données et comportement

L’état est la situation actuelle du système : utilisateur connecté, commande payée, fichier traité. Les données sont la représentation durable de cet état. Le comportement est la règle qui fait passer d’un état à un autre. Beaucoup de bugs viennent d’une transition autorisée au mauvais moment ou répétée deux fois.

> Question d’architecte
> Quelles données entrent ? Qui peut les modifier ? Quelle règle s’applique ? Quel résultat doit être observable ? Que se passe-t-il si l’opération est répétée ou interrompue ?

Les frontières à rendre visibles

Une frontière sépare deux responsabilités : navigateur et serveur, application et base, service et fournisseur externe. À chaque frontière, exige un contrat : données attendues, réponses possibles, erreurs, délai, sécurité. Une interface floue déplace les bugs au lieu de les résoudre.

Couplage et cohésion

La cohésion mesure si un module fait une chose qui a du sens. Le couplage mesure combien il dépend des autres. Vise une forte cohésion et un couplage faible : le module Facturation connaît les factures, mais ne devrait pas décider du dessin des boutons ni connaître les détails du serveur de courriel.

CHAPITRE 2

Architecture : séparer pour maîtriser

Une bonne architecture n’est pas spectaculaire. Elle rend le changement local, la panne compréhensible et la décision réversible.

Les couches utiles

1. Présentation
   Ce que l’utilisateur voit et manipule.
2. Application
   Les scénarios : créer une commande, valider un devis, exporter un PDF.
3. Domaine
   Les règles qui donnent son sens au produit.
4. Infrastructure
   Base, fichiers, courriels, paiements, services externes.

Cette séparation n’impose pas quatre serveurs ni un framework complexe. Elle peut exister dans un monolithe simple, sous forme de dossiers et de dépendances orientées proprement.

Monolithe modulaire d’abord

Pour la plupart des projets métier, commence par une seule application déployable, découpée en modules cohérents. Les microservices ajoutent réseau, authentification interservices, observation distribuée et déploiements multiples. Ils résolvent surtout des problèmes d’échelle organisationnelle déjà prouvés.

Décision par défaut
Une base de code, un déploiement, des modules clairs. Extraire un service seulement lorsqu’une contrainte mesurée le justifie : charge indépendante, sécurité particulière ou équipe autonome.

API, contrat et idempotence

Une API est une frontière programmable. Son contrat doit préciser la requête, la réponse, les codes d’erreur et les droits. Une opération idempotente produit le même résultat si elle est répétée : indispensable pour les paiements, webhooks et reprises après coupure.

Dépendances et dette

Une dépendance économise du travail mais crée une obligation : mises à jour, vulnérabilités, changement de licence, disparition du projet. Demande à l’agent pourquoi elle est nécessaire, quelle alternative standard existe et comment la remplacer. La dette technique n’est pas du mauvais code en général ; c’est un raccourci dont le coût futur est connu ou ignoré.

Les qualités non fonctionnelles

Une fonctionnalité dit ce que fait le système. Les qualités non fonctionnelles disent comment il doit le faire : sécurité, performance, disponibilité, accessibilité, maintenabilité, souveraineté des données. Elles doivent devenir des critères mesurables, pas des adjectifs.

CHAPITRE 3

Concevoir avant de coder

Le meilleur moyen d’aller vite consiste à réduire l’ambiguïté avant qu’elle ne se transforme en fichiers, tables et dépendances.

Du problème à la tranche verticale

Décris d’abord le résultat humain : « un client dépose un fichier et reçoit une confirmation ». Une tranche verticale traverse juste assez d’interface, logique et stockage pour produire ce résultat. Elle est démontrable et testable. Évite les grands chantiers horizontaux du type « faire toute la base », puis « faire toute l’interface ».

La fiche de fonctionnalité

1. Intention
   Pour qui ? Quel problème ? Quelle amélioration observable ?
2. Scénario nominal
   Le chemin simple, étape par étape.
3. Règles
   Ce qui est autorisé, interdit, calculé ou conservé.
4. Cas limites
   Données absentes, doublons, coupure, lenteur, droits insuffisants.
5. Critères d’acceptation
   Des phrases vérifiables qui définissent terminé.
6. Hors périmètre
   Ce que cette tranche ne tente pas de résoudre.

ADR : enregistrer les décisions

Un Architecture Decision Record tient souvent sur une page : contexte, décision, options rejetées, conséquences. Il évite que l’agent ou un futur collaborateur redécouvre l’histoire en lisant le code. Exemples : choix de PostgreSQL, stockage S3, monolithe modulaire, stratégie d’authentification.

> Bon critère d’acceptation
> « Après un double clic sur Envoyer, une seule commande est créée et le bouton affiche l’état en cours » est vérifiable. « Le formulaire fonctionne bien » ne l’est pas.

Modèle de données : nommer le réel

Les tables et objets sont un vocabulaire du métier. Choisis des noms stables et non ceux de l’écran du moment. Décide les identifiants, relations, contraintes d’unicité et règles de suppression. Toute modification de structure doit passer par une migration versionnée.

CHAPITRE 4

Le harnais agentique

Le harnais est le système qui transforme un modèle génératif rapide en collaborateur contraint, informé et vérifiable.

Ce que contient un bon harnais

1. Carte du dépôt
   Où sont l’interface, le domaine, les tests, les scripts et la documentation.
2. Règles permanentes
   Conventions, commandes autorisées, sécurité, responsabilités des modules.
3. Contexte de tâche
   Problème, preuves, fichiers concernés, critères d’acceptation.
4. Boucle de preuve
   Tests, analyse statique, revue du diff, exécution réelle.
5. Garde-fous
   Pas de secret dans Git, pas de destruction, pas de déploiement implicite.
6. Mémoire durable
   ADR, documentation, journal de versions et procédures.

Demander une enquête avant une modification

Une bonne mission commence par inspecter. Demande : reproduis le problème, identifie la cause, propose le changement minimal, liste les risques et attends ou exécute selon l'autorisation donnée. Une erreur courante est de confondre le symptôme visible avec la cause réelle.

Un prompt opératoire

```
Contexte : ce projet fait…
Objectif : obtenir…
Contraintes : préserver…, ne pas…
Acceptation : étant donné…, quand…, alors…
Méthode : inspecte, propose, modifie par petites étapes, teste, relis le
diff.
Livraison : résume les fichiers, preuves, risques restants et procédure de
retour arrière.
```

Les rôles utiles

Le Pilote découpe et arbitre. Le Bâtisseur implémente. Le Testeur cherche les échecs. Le Critique challenge la conception. Le Gardien vérifie sécurité et règles. L'Éclaireur recherche les options. L'Archiviste maintient les traces. L'Analyste mesure. Ces rôles peuvent être joués successivement par un agent ; l'indépendance de la revue compte davantage que le nombre d'agents.

> Règle d'or
> Ne demande jamais seulement « fais la fonctionnalité ». Demande aussi quelles preuves montreront qu'elle fonctionne et quelles conséquences elle crée.

CHAPITRE 5

L’ordre juste pour construire

Coder simplement, c’est obtenir tôt une preuve complète, puis élargir sans perdre la maîtrise.

1	Observer Lire le dépôt, exécuter l’existant, reproduire le besoin ou le bug.
2	Définir Écrire scénario, critères, limites et risque principal.
3	Dessiner Choisir le flux, les responsabilités et le modèle de données minimal.
4	Préparer Créer une branche, vérifier l’état Git, identifier les commandes de test.
5	Prouver Écrire un test ou une reproduction qui échoue pour la bonne raison.
6	Implémenter Faire le plus petit changement cohérent.
7	Vérifier Tests ciblés, suite complète, analyse statique et essai humain.
8	Relire Inspecter le diff : intention, bruit, secrets, migrations, dépendances.

9	Documenter Mettre à jour ADR, README, configuration et journal si nécessaire.
10	Publier Commit clair, push, PR, CI, merge, déploiement progressif, observation.

La boucle rouge, verte, propre

Rouge : une preuve échoue et démontre le manque. Vert : le changement minimal la fait passer. Propre : on simplifie sans changer le comportement. Même sans test-first strict, cette logique évite d’écrire un test qui ne fait que confirmer son propre code.

Petit ne veut pas dire morcelé

Un changement doit être petit mais complet : une intention, une preuve, une conséquence lisible. Dix micro-commits incompréhensibles ne sont pas supérieurs à un commit cohérent. La taille se juge par la capacité à relire et revenir en arrière.

Definition of Done

> Terminé signifie
> Critères satisfaits ; tests pertinents réussis ; erreurs traitées ; logs utiles ; sécurité examinée ; documentation à jour ; migration et retour arrière prévus ; diff relu ; déploiement observable.

CHAPITRE 6

Git sans folklore

Git est une machine à enregistrer des états et à organiser la collaboration. Il ne remplace ni la sauvegarde des données ni le déploiement.

Le modèle mental

Le dossier de travail contient tes modifications. La zone de préparation sélectionne ce qui entrera dans le prochain commit. Le commit est un instantané nommé avec un parent. La branche est une étiquette mobile pointant vers une suite de commits. Le dépôt distant permet le partage et la sauvegarde de l’histoire du code.

Les verbes essentiels

Commande	Sens
status	Voir la situation avant toute action.
diff	Lire les changements non préparés.
add	Choisir ce qui entrera dans le commit.
commit	Enregistrer un état cohérent localement.
push	Envoyer les commits vers le dépôt distant.
pull	Récupérer et intégrer des changements distants.
fetch	Récupérer sans intégrer.
merge	Réunir deux histoires.
revert	Créer un commit qui annule un ancien changement.

Qu’est-ce qu’une PR ?

Une Pull Request - PR - est une proposition d’intégrer une branche dans une autre. Elle montre le diff, porte la discussion, déclenche la CI et conserve l’accord. Utilise-la pour tout changement de production qui mérite une seconde lecture, touche aux données, à la sécurité, à l’architecture ou dépasse une correction triviale.

> Une PR n’est pas
> Un synonyme de push, un déploiement, ni une preuve automatique de qualité. Elle est un point de contrôle social et technique.

Commit, push, PR, merge : la séquence

1. Commit local
   Instantané cohérent et message expliquant pourquoi.
2. Push
   Publication de la branche sur le serveur Git.
3. PR
   Comparaison, explication, tests automatiques et revue.
4. Merge
   Intégration dans la branche de référence après accord.
5. Tag ou release
   Nom stable donné à la version publiable.
6. Déploiement
   Installation de cette version dans un environnement.

Messages utiles

```
feat(commandes): empêcher la création en double
fix(upload): reprendre après expiration du lien
docs(architecture): consigner le choix du stockage objet
```

CHAPITRE 7

Tests, revue et preuves

Un test n’est pas une cérémonie. C’est une affirmation exécutable sur un comportement important.

La pyramide pragmatique

Les tests unitaires vérifient vite une règle isolée. Les tests d’intégration vérifient la coopération avec la base, le système de fichiers ou une API. Les tests de bout en bout parcourent un scénario utilisateur réel, plus coûteux et plus fragile. Ajoute des tests contractuels aux frontières critiques.

Que tester en priorité ?

1. Règles métier
   Calculs, transitions, permissions, unicité.
2. Frontières
   Base, stockage, paiement, courriel, API.
3. Échecs coûteux
   Perte, duplication, divulgation, blocage.
4. Régressions
   Chaque bug corrigé reçoit une preuve qui aurait dû le détecter.
5. Parcours vitaux
   Connexion, commande, paiement, export, restauration.

CI : l’atelier automatique

L’intégration continue exécute sur chaque push ou PR une recette propre : installer les dépendances verrouillées, analyser le code, lancer les tests, construire l’artefact. Si la CI échoue, la PR ne doit normalement pas être fusionnée. Le vert prouve uniquement ce qui a été testé.

Relire un diff

Lis le diff comme une décision : chaque fichier sert-il l’objectif ? Une dépendance ou une migration apparaît-elle ? Une erreur est-elle masquée ? Des logs contiennent-ils des données sensibles ? Les tests échoueraient-ils si on retirait le correctif ? Le changement est-il réversible ?

> Preuve minimale avant publication
> Suite automatique verte + test du scénario réel + revue du diff + sauvegarde ou retour arrière approprié + personne clairement responsable de l’observation.

Revue agentique indépendante

Après l’implémentation, donne à une session fraîche l’objectif, le diff et les critères, sans lui demander de défendre la solution. Sa mission : chercher les hypothèses cachées, risques, cas limites et tests manquants. L’agent bâtisseur corrige ensuite ; le décideur arbitre.

CHAPITRE 8

Du local au VPS

Publier n’est pas copier des fichiers au hasard : c’est promouvoir une version identifiée à travers des environnements maîtrisés.

Les environnements

1. Local
   Rapide, isolé, données factices, outils de développement.
2. CI
   Machine propre qui reproduit les vérifications.
3. Staging
   Configuration proche de la production pour essais réalistes.
4. Production
   Service réel, données réelles, accès contrôlé et observation continue.

Le pipeline recommandé

1	Coder sur une branche Tranche limitée et critères explicites.
2	Tester localement Tests ciblés puis suite pertinente.
3	Commit et push Version identifiable sur le dépôt distant.
4	Ouvrir la PR Description, captures, risques, migration, rollback.
5	Laisser la CI prouver Tests, lint, sécurité, construction.
6	Faire relire et merger Accord humain ou politique définie.
7	Construire un artefact Image Docker ou paquet immuable lié au commit.
8	Déployer en staging Migration, smoke tests, parcours critique.
9	Promouvoir en production Même artefact, variables et secrets de production.
10	Observer Santé, erreurs, latence, parcours métier, puis décision de poursuivre ou revenir.

Ce qui vit sur le VPS

Le reverse proxy reçoit HTTPS et route vers l’application. L’application s’exécute comme service ou conteneur. La base persiste sur un volume protégé. Les secrets viennent de l’environnement ou d’un coffre, jamais du dépôt. Les journaux et métriques quittent idéalement la machine ou sont conservés avec rotation.

Migration de base

Une migration est du code versionné qui transforme le schéma. Sauvegarde avant une transformation risquée. Préfère les changements compatibles en plusieurs temps : ajouter, remplir, basculer, puis supprimer plus tard. Une ancienne version doit pouvoir cohabiter pendant le déploiement si le trafic continue.

Smoke test et contrôle de santé

Le health check dit que le processus répond. Le smoke test confirme quelques fonctions vitales : page accessible, connexion base, authentification, création contrôlée, file de tâches. Il ne remplace pas la suite de tests.

> La phrase à exiger
> « La production exécute l’artefact construit depuis le commit X ; la migration Y a réussi ; les smoke tests Z sont verts ; le rollback consiste à… »

CHAPITRE 9

Optimiser sans abîmer

L’optimisation commence par une mesure, vise un goulot réel et conserve une preuve de comportement.

Les trois coûts

Le temps machine : CPU, mémoire, disque, réseau. Le temps humain : compréhension, modification, diagnostic. Le risque : panne, corruption, sécurité. Un code plus court mais opaque peut économiser des millisecondes et coûter des journées.

La méthode de performance

1. Définir
   Quel parcours est trop lent ? Quel objectif chiffré ?
2. Mesurer
   Profilage, requêtes, traces, percentile p95 plutôt qu’impression.
3. Localiser
   Trouver le goulot dominant.
4. Changer
   Une hypothèse et une modification à la fois.
5. Comparer
   Même charge, avant/après, comportement inchangé.
6. Surveiller
   Vérifier en production et prévenir la régression.

Les gains fréquents

Éviter les requêtes répétées, ajouter un index justifié, paginer, réduire les données transférées, mettre en cache une donnée stable, déplacer un travail long en tâche asynchrone, compresser les images, supprimer une dépendance lourde. Ne cache pas avant d’avoir compris l’invalidation.

Optimiser le code pour l’humain

Noms précis, fonctions courtes autour d’une intention, erreurs explicites, peu d’état global, dépendances injectées, commentaires sur le pourquoi, documentation proche du code. La duplication locale peut être moins coûteuse qu’une abstraction prématurée.

> Signal d’alarme
> Si l’agent propose microservices, cache distribué, file de messages et orchestration sans métriques ni contrainte prouvée, demande le problème exact que chaque élément résout.

CHAPITRE 10

Incidents, sauvegardes et retour arrière

Un système professionnel ne promet pas l’absence de panne. Il réduit l’impact, détecte vite et sait restaurer.

Rollback et roll-forward

Le rollback remet l’ancienne version. Le roll-forward publie un correctif. Une migration destructive peut rendre le rollback impossible ; d’où les migrations compatibles et les sauvegardes testées. Décide avant le déploiement quel signal déclenche l’arrêt.

Sauvegarder n’est pas restaurer

Une sauvegarde utile est automatique, chiffrée, datée, conservée hors du VPS et régulièrement restaurée sur un environnement de test. Définis le RPO - quantité maximale de données acceptables à perdre - et le RTO - temps maximal acceptable pour restaurer le service.

La procédure d’incident

1. Détecter
   Alerte ou signal utilisateur qualifié.
2. Stabiliser
   Arrêter l’aggravation : désactiver, isoler, revenir.
3. Communiquer
   Impact, responsable, prochaine mise à jour.
4. Diagnostiquer
   Chronologie, logs, métriques, changements récents.
5. Rétablir
   Retour arrière, restauration ou correctif vérifié.
6. Apprendre
   Post-mortem sans blâme, actions datées et test de non-régression.

Observabilité

Les logs racontent des événements. Les métriques quantifient dans le temps. Les traces suivent une requête à travers les composants. Ajoute des identifiants de corrélation, des niveaux de logs et des alertes liées à l’expérience réelle, pas seulement au CPU.

Avant une opération risquée
Sauvegarde vérifiée, version actuelle identifiée, commande exacte, cible exacte, temps estimé, critère de succès, personne disponible et procédure de retour.

CHAPITRE 11

La méthode ORCHESTRE

Une méthode courte à appliquer à chaque fonctionnalité, correction ou publication.

O	Observer Le système réel, ses contraintes et les preuves.
R	Résultat Formuler la valeur et les critères d’acceptation.
C	Cartographier Données, flux, frontières, risques et propriétaires.
H	Hypothèse Choisir la solution minimale et ses conséquences.
E	Expérimenter Créer une preuve qui échoue, puis implémenter.
S	Sécuriser Tests, droits, erreurs, sauvegarde, cas limites.
T	Tracer Commit, ADR, documentation et journal de version.
R	Relire Diff et revue indépendante.

E Exposer
Staging, production progressive, observation, rollback.

Le point d’arrêt décisionnel

Après Cartographier et Hypothèse, arrête l’agent si la solution crée une nouvelle dépendance majeure, modifie les données, touche à l’authentification, engage un fournisseur ou change l’architecture. Tu arbitres alors avec un ADR court. Le reste peut suivre automatiquement dans le périmètre autorisé.

Le compte rendu idéal de l’agent

Livraison
Résultat obtenu ; fichiers et décisions ; tests exécutés avec résultats ; éléments non testés ; risques restants ; migration éventuelle ; commandes de publication ; méthode de retour arrière.

Ta responsabilité

Tu n’as pas à vérifier chaque caractère. Tu dois vérifier l’alignement entre problème, décision, preuve et risque. Quand l’un manque, le travail n’est pas prêt. L’agent est responsable de montrer son raisonnement opératoire ; toi, de décider ce qui mérite confiance.

CHAPITRE 12

Fiches réflexes et glossaire

Les pages à ouvrir quand tu pilotes une session, une PR ou une mise en production.

Fiche 1 - Lancer une tâche

1. Donner le contexte
   But du produit, utilisateurs et zone concernée.
2. Dire le résultat
   Comportement observable, pas technologie imposée.
3. Fixer les limites
   Données, sécurité, fichiers, opérations interdites.
4. Définir les preuves
   Critères, tests, capture ou démonstration.
5. Exiger le retour
   Diff, décisions, risques et suite.

Fiche 2 - Accepter une PR

```
□ Le besoin est expliqué. □ Le diff est limité. □ La CI est verte. □ Les tests couvrent le risque. □ Les erreurs sont visibles. □ Aucun secret. □ Migration compatible. □ Documentation à jour. □ Rollback crédible.
```

Fiche 3 - Autoriser la production

```
□ Commit/tag identifié. □ Sauvegarde appropriée. □ Artefact immuable. □ Variables et secrets présents. □ Migration répétée en staging. □ Smoke tests définis. □ Observateur nommé. □ Seuil d’arrêt et rollback prêts.
```

Glossaire essentiel

API — Contrat permettant à deux logiciels d’échanger.
Artefact — Résultat construit et déployable lié à une version.
Branche — Pointeur vers une ligne de commits.
Build — Transformation du code source en résultat exécutable.
CI/CD — Vérification continue / livraison ou déploiement automatisé.
Commit — Instantané versionné accompagné d’un message.
Conteneur — Paquet isolé regroupant application et environnement d’exécution.
Déploiement — Mise en service d’une version dans un environnement.
Diff — Comparaison précise entre deux états du code.
Endpoint — Adresse et opération exposées par une API.
Framework — Cadre logiciel structurant l’application.
Idempotence — Propriété d’une opération répétable sans effet supplémentaire.
Lint — Analyse automatique de conventions et erreurs probables.
Merge — Réunion de deux lignes d’histoire Git.
Migration — Transformation versionnée du schéma ou des données.
PR — Proposition documentée d’intégrer une branche.
Refactorisation — Amélioration interne sans changer le comportement.
Régression — Fonctionnement ancien cassé par un changement.
Rollback — Retour à une version antérieure.
Secret — Information sensible nécessaire à l’exécution.
Staging — Environnement de répétition proche de la production.
VPS — Serveur virtuel administré sur lequel tournent les services.
Webhook — Appel automatique envoyé lors d’un événement.

Dernière page

La simplicité est une discipline

Un logiciel simple n’est pas un logiciel naïf. C’est un système dont les responsabilités sont visibles, les décisions tracées, les risques proportionnés et les changements vérifiables. L’agent peut accélérer chaque geste. La méthode conserve la direction.

À garder en tête
Une intention par changement. Une preuve par risque. Une trace par décision. Un retour possible par publication.

Fin de la première édition. Ce manuel est conçu pour être annoté, challengé et transformé en méthode vivante.