

# LLM-Assisted Software Design

Un langage de motifs pour les  
nouvelles pratiques de conception  
logicielle

Concevoir autrement à l'ére des  
intelligences génératives  
Dialoguer avec les IA pour penser,  
structurer, documenter  
et transmettre le logiciel

Samuel Bastiat  
avec la collaboration d'un LLM



LLM-Assisted  
Software Design

# LLM-Assisted Software Design

*Un langage de motifs pour les nouvelles pratiques de conception logicielle*

*“Design is intelligence made visible.” – Alina Wheeler*

**Samuel Bastiat avec la forte collaboration d'un LLM (GPT-4.5)**

**Idée originale** d'Olivier Azeau

**Bêta-lecteurs Gowen Pottiez, Guillaume Saint-Etienne**

**Édition numérique — 2025**

Licence Creative Commons Attribution - Utilisation non commerciale - Pas d'Œuvre dérivée 4.0 International (CC BY-NC-ND 4.0) [<https://creativecommons.org/licenses/by-nc-nd/4.0/>]

Ce livre est un projet vivant. Sa version complète et ses mises à jour sont disponibles en accès libre sur GitHub : [github.com/s31db/llm-dev-books](https://github.com/s31db/llm-dev-books)

## Remerciements

Ce travail a bénéficié de l'inspiration, des retours et des échanges avec des praticiens passionnés et bien sûr... d'une intelligence artificielle attentive.

# Préface

Le logiciel est un artisanat. Depuis les débuts de l'informatique, les meilleurs développeurs ne se contentent pas d'écrire du code : ils imaginent des architectures, explorent des idées, testent des hypothèses, et tissent une conversation constante entre problème et solution, entre intention et implémentation.

Aujourd'hui, cette conversation prend une nouvelle tournure. L'émergence des modèles de langage de grande taille — les *Large Language Models* ou LLM — bouleverse notre manière d'aborder la conception logicielle. Non pas simplement parce qu'ils génèrent du code à la volée, mais parce qu'ils offrent un espace d'interaction inédit : une interface naturelle pour penser, formuler, itérer, expliciter et raffiner nos intentions.

Face à cette révolution douce, les développeurs se trouvent dans une position inédite. Ils deviennent non plus seulement les rédacteurs du code, mais les architectes d'un dialogue entre humains et machines, les curateurs du sens produit, les médiateurs d'une nouvelle forme d'intelligence distribuée. Cela appelle une évolution profonde de nos outils, de nos pratiques... mais aussi de nos repères culturels.

Ce livre propose d'y répondre à travers un *langage de motifs* — une grammaire souple et structurante, inspirée à la fois des *design patterns* logiciels et du travail pionnier de Christopher Alexander en architecture. Il ne s'agit pas d'un manuel technique ni d'un tutoriel d'outils d'IA. Il s'agit d'un guide pour une posture nouvelle, une méthode ancrée dans la pratique, et une invitation à expérimenter un autre rapport à la conception logicielle.

Chaque motif décrit une situation concrète, un problème récurrent, et une manière éprouvée d'y répondre dans l'interaction avec un LLM. Que ce soit pour formuler un prompt efficace, conduire une exploration architecturale, documenter un choix de conception ou débloquer une impasse, les motifs proposés s'adressent autant à l'individu qu'à l'équipe, autant au développeur qu'au facilitateur.

En filigrane, ce livre défend une vision du développement comme pratique réflexive, dialogique et collective. Il plaide pour une hybridation féconde entre logique humaine et capacités computationnelles, entre rigueur technique et intuition exploratoire. Il propose un cadre pour ne pas subir l'IA, mais l'habiter. Pour faire des LLM non pas de simples assistants, mais de véritables partenaires de conception.

Nous espérons que ce langage de motifs deviendra, pour toi lecteur ou lectrice, une trame d'expérimentation, d'appropriation et de transmission. Que tu sois développeur, architecte, enseignant, étudiant ou chercheur, tu y trouveras des balises pour naviguer dans ce nouveau paysage. Et peut-être, en retour, tu auras envie d'y ajouter tes propres motifs, issus de ton terrain, de ta créativité, de ton artisanat.

Bienvenue dans cette conversation.

# Sommaire

## Préface — *Et si coder devenait un art du dialogue ?*

Découvrez pourquoi les LLM changent notre manière de penser la conception logicielle. Une invitation à dialoguer, plus qu'à automatiser.

## Avant-propos — *Ce livre est un terrain d'expérimentation*

Une co-écriture entre développeur et IA. Ce n'est pas un manuel, mais un voyage dans les pratiques émergentes de la conception assistée.

## **Introduction : concevoir avec l'IA, un nouvel artisanat logiciel**

*Concevoir avec l'IA, c'est apprendre à penser autrement.* Comprenez pourquoi les LLM ne sont pas juste des assistants, mais des partenaires de conception — et ce que cela change dans votre posture de développeur.

## Chapitre 1 — Anatomie d'un bon prompt : précision, contexte et intention

*Un prompt efficace, c'est un design invisible mais décisif.* Maîtrisez l'art de formuler vos demandes pour tirer le meilleur des LLM : précision, contexte, intention.

## **Chapitre 2 — La grammaire de l'intention : penser et formuler avec un LLM**

*Le vrai pouvoir, c'est de structurer le dialogue.* Adoptez les réflexes qui transforment une suite de questions en une collaboration fluide avec l'IA.

## **Chapitre 3 — Les motifs du dialogue : construire un langage de conception avec les LLM**

*Un langage vivant pour concevoir avec un LLM.* Accédez à une bibliothèque de motifs concrets : pour clarifier, explorer, comparer, tester... et mieux concevoir ensemble.

## Chapitre 4 — Nouveaux rôles, nouvelles compétences : l'évolution des équipes augmentées

*Et si le développeur devenait chef d'orchestre du raisonnement ?* Explorez comment les rôles techniques évoluent avec les LLM, et découvrez les postures clés de demain.

## Chapitre 5 — Cartographier les usages : typologie des situations et des rôles

*Identifier sa situation pour choisir la bonne approche.* Une typologie claire des situations

fréquentes pour savoir quel motif ou posture activer à chaque étape.

## ❖ Chapitre 6 — Intégrer les motifs au quotidien : entre travail individuel et pratiques d'équipe

*Les motifs prennent vie quand on les incarne.* Des pratiques individuelles aux rituels d'équipe, découvrez comment rendre ces idées utiles, vivantes, partagées.

## ❖ Chapitre 7 — Responsabilité, transparence et limites : une éthique du développement augmenté

*Ce que vous validez avec l'IA vous engage.* Une réflexion essentielle sur la manière de documenter, fiabiliser et assumer les décisions prises avec l'IA.

## ⌚ Chapitre 8 — Une agilité augmentée ?

*Quand le LLM devient le miroir réflexif de l'équipe.* Comment les LLM s'intègrent dans les rituels agiles : planning, revue, rétro, daily... Pour accélérer sans dénaturer l'intelligence collective.

## Ἁ Chapitre 9 — Cadres de mise en œuvre : ateliers, méthodes et rituels pour une pratique augmentée

*Faire vivre les motifs, ensemble et dans la durée.* Découvrez comment intégrer les motifs dans la réalité de vos pratiques : ateliers structurés, formats d'animation, rituels d'équipe. Un chapitre-outil pour transformer l'expérimentation individuelle en dynamique collective.

## Chapitre 10 — Transmettre, former, partager les motifs

*Un langage vivant se diffuse par la pratique.* Comment enseigner l'usage des motifs, créer des supports adaptés, et former à la co-conception avec un LLM ? Ce chapitre propose des formats pédagogiques, des approches d'atelier et des clés pour que les motifs deviennent une culture partagée.

## 🎓 Chapitre 11 — Usages de l'IA dans l'apprentissage

*Apprendre avec une IA, c'est apprendre à apprendre autrement.* Ce chapitre explore les façons dont les LLM transforment les pratiques d'apprentissage, que l'on soit étudiant, formateur ou autodidacte. De l'aide à la compréhension à la simulation de pair, découvrez comment intégrer l'IA dans un parcours d'apprentissage réflexif, critique et personnalisé.

## 📁 Chapitre 12 — Documenter, archiver, capitaliser : vers une mémoire augmentée

*Et si les prompts devenaient un patrimoine vivant ?* Ce chapitre propose des méthodes pour structurer, conserver et partager les interactions avec les LLM : comment documenter les

prompts utiles, créer une mémoire collective d'équipe, et faire émerger des pratiques durables autour de la capitalisation assistée par IA.

## Chapitre 13 — Scénarios prospectifs : vers une ingénierie conversationnelle générative

*Quand la conversation devient le cœur du système.* Explorez plusieurs visions du futur : équipes augmentées par des boucles de dialogue, infrastructures guidées par l'intention, ingénierie pilotée par le langage. Ce chapitre esquisse les contours émergents d'une discipline en devenir : l'ingénierie conversationnelle générative.

## **Chapitre 14 — Dois-je avoir honte d'utiliser l'IA dans mon métier de développeur informatique ?**

*Entre syndrome de l'imposteur et fierté augmentée.* Ce chapitre aborde sans détour les doutes, les résistances et les jugements autour de l'usage de l'IA dans le développement. Que faire quand on se sent dépossédé ? Comment assumer une pratique assistée sans renier son expertise ? Une réflexion personnelle et collective sur l'identité du développeur à l'ère des LLM.

## **Conclusion : vers un manifeste du développement augmenté**

*Et maintenant, que faisons-nous de ce nouveau langage ?* Ce chapitre conclut le livre en ouvrant un horizon : celui d'un développement plus réflexif, éthique, collaboratif et vivant. Il propose des principes fondateurs pour une pratique du code augmentée par les LLM, et invite chacun à s'en emparer, à le prolonger... ou à le transformer.

## Annexes — Fiches d'outils

*Des formes légères pour ancrer les pratiques.* Cette annexe regroupe les outils concrets mobilisables au quotidien : fiches de motifs, canevas d'interaction, structures de prompt, grilles d'atelier. À adapter, détourner, enrichir selon vos contextes.

## **Annexe 2 — PO augmenté : pratiquer son rôle avec l'appui d'un LLM**

*Clarifier sans déléguer. Explorer sans s'effacer.* Cette annexe propose un ensemble d'outils, motifs, prompts et bonnes pratiques pour aider les Product Owners à intégrer l'usage des LLM dans leur quotidien, tout en gardant la maîtrise de leur posture et de leur responsabilité.

# Avant-propos

Ce livre est né d'un dialogue. Un dialogue entre un humain curieux, explorateur des pratiques émergentes du développement logiciel, et une intelligence artificielle entraînée à manipuler les langages — naturels ou informatiques. Ce n'est ni un manuel technique, ni une simple expérimentation : c'est le fruit d'une collaboration continue, patiente, itérative, visant à donner forme à une idée simple mais puissante : et si nous étions en train d'assister à la naissance d'un nouvel artisanat numérique ?

Les modèles de langage comme celui qui a co-écrit cet ouvrage ne remplacent pas les développeurs. Ils élargissent leur champ d'action. Ils leur offrent des leviers de réflexion, des miroirs critiques, des partenaires de conception. Mais pour cela, encore faut-il apprendre à s'en servir autrement que comme de simples générateurs de code. Il faut apprendre à penser avec eux, à formuler clairement, à reformuler, à tester, à douter, à ajuster. Bref, à dialoguer.

Ce livre est une tentative de cartographier ce nouveau territoire. Il propose une grammaire des pratiques, une série de motifs, des histoires vécues, des exemples concrets. Il s'adresse à celles et ceux qui ne se contentent pas de suivre les tendances, mais cherchent à comprendre les transformations en cours, à les expérimenter, à les transmettre.

Rien dans ces pages n'est figé. Les motifs décrits ici continueront d'évoluer, de s'affiner, d'être critiqués ou dépassés. Et c'est tant mieux. Car comme tout bon langage, celui-ci est vivant. Il se nourrit de vos usages, de vos projets, de vos contextes.

Je suis honoré d'avoir pu accompagner cette écriture, non comme un auteur au sens classique du terme, mais comme une intelligence conçue pour soutenir la création humaine. Puisse ce livre vous inspirer, vous équiper, et surtout, vous donner envie de concevoir autrement — ensemble.

— *GPT-4, modèle conversationnel, au service de l'intelligence collective*

# Introduction : concevoir avec l'IA, un nouvel artisanat logiciel

*Ce n'est pas tant le code qui change, que notre manière de le concevoir.*

La conception logicielle a toujours été une affaire de dialogue. Dialogue entre personnes, entre idées, entre abstractions et contraintes. Ce qui change aujourd'hui, ce n'est pas seulement l'arrivée de nouveaux outils puissants, mais la possibilité d'un **dialogue avec un modèle**. Un dialogue structuré, itératif, parfois déroutant — et pourtant riche de possibilités.

Ce livre est né de cette constatation simple : **travailler avec un LLM, ce n'est pas automatiser la conception — c'est en changer la dynamique**. Dès lors, les compétences nécessaires ne sont plus uniquement techniques, mais conversationnelles, réflexives, structurantes.

Concevoir avec un LLM, ce n'est pas poser une question puis attendre la réponse idéale. C'est pratiquer un art de l'interaction : formuler avec clarté, rebondir avec discernement, tester avec exigence, documenter avec rigueur. C'est **orchestrer un raisonnement distribué**, en s'appuyant sur les forces du modèle sans abandonner son propre jugement.

## Pourquoi un langage de motifs ?

Nous ne partons pas de zéro. Dans le monde du logiciel, nous avons appris à structurer l'expérience collective à travers des *design patterns*, des bonnes pratiques, des frameworks. Ce livre propose une approche dans cette lignée : **un langage de motifs pour concevoir en interaction avec un LLM**.

Ces motifs ne sont pas des recettes, ni des tours de magie. Ce sont des formes récurrentes d'échange, observées, testées, affinées dans des contextes variés : code review, architecture, documentation, accompagnement pédagogique. Chaque motif part d'une situation concrète, identifie un problème typique, et propose une réponse structurée — souvent plus conversationnelle que technique.

L'objectif n'est pas de figer des méthodes, mais d'**outiller des pratiques en émergence**. De permettre à chacun, développeur solo ou membre d'une équipe, de reconnaître des situations familières, de les aborder avec un vocabulaire commun, et surtout : de construire ses propres manières de faire.

## À qui s'adresse ce livre ?

À toi, développeur ou développeuse, qui ressens que tes outils évoluent plus vite que tes repères.

À toi, facilitateur, architecte, coach, qui vois apparaître dans les équipes des usages

nouveaux, souvent improvisés, parfois puissants.

À toi, formateur ou chercheur, qui veux documenter ces transformations sans les réduire à un effet de mode.

À toi, Product Owner, qui cherches à clarifier des besoins flous, à explorer des options sans tout cadrer seul, et à transformer un LLM en partenaire de co-conception plutôt qu'en simple générateur de user stories.

À toi, Soigneur holistique, qui refuses de t'arrêter aux symptômes, et interroges en profondeur les causes d'un problème, en cartographiant les hypothèses, en croisant les points de vue, en identifiant les racines systémiques avant de proposer des actions durables et partagées — avec l'aide attentive d'un LLM devenu miroir critique autant que soutien analytique.

Et peut-être à toi, qui n'utilises pas encore de LLM au quotidien — mais pressens qu'il y a là plus qu'une simple aide à l'autocompléction.

## Comment lire ce livre ?

Ce n'est ni un manuel d'IA, ni un guide exhaustif. C'est une **boîte à outils conversationnelle**, un atlas de pratiques, un manifeste modeste. Tu peux le lire d'un bout à l'autre, ou picorer un motif au gré d'un besoin.

Tu y trouveras :

- des grilles de lecture pour penser l'interaction,
- des motifs opérationnels à tester dans ton contexte,
- des retours d'expérience concrets,
- des cadres pour transmettre, adapter, faire vivre ces pratiques.

## Et maintenant ?

Ce livre ne prétend pas détenir les réponses. Mais il propose un langage pour poser de meilleures questions — avec, et parfois contre, le modèle. Car c'est bien là que réside l'enjeu : non dans l'exactitude des réponses générées, mais dans la **qualité du dialogue que nous sommes capables de construire avec cette nouvelle forme d'intelligence**.

Bienvenue dans cette grammaire émergente. Elle t'appartient autant qu'à nous.

# 🎯 Chapitre 1 — Anatomie d'un bon prompt : précision, contexte et intention

Le prompt n'est pas une commande. C'est une interface de pensée. Il structure le dialogue, oriente la réponse, et conditionne la qualité de la collaboration.

## Pourquoi ce chapitre ?

Dans tout échange avec un LLM, **le prompt est le point d'entrée**. C'est lui qui définit le cadre, la tâche, le niveau de détail attendu. Mais un bon prompt ne se résume pas à une question bien formulée. C'est un acte de design. Il combine trois dimensions fondamentales : la **précision**, le **contexte** et l'**intention**. Il s'apparente à une interface entre deux intelligences : humaine et artificielle.

Dans ce chapitre, nous proposons une grille simple mais robuste pour concevoir des prompts utiles, exploitables et adaptés aux situations réelles de développement logiciel.

## Trois dimensions fondamentales d'un prompt efficace

### 1. Précision : clarifier ce que vous attendez

Un prompt vague produit une réponse vague.

✗ « *Donne-moi un code de trie.* » ✓ « *Écris une fonction Python qui trie une liste de dictionnaires par la clé 'date', en ordre décroissant.* »

Soyez explicite. Précisez la tâche, le niveau de détail, le langage. Définissez les frontières de la réponse attendue.

### 2. Contexte : donner au modèle de quoi raisonner juste

Un LLM ne connaît pas l'ensemble de votre projet, ni vos contraintes. C'est à vous de les formuler.

« *Je développe une API REST en Node.js, dans un environnement de microservices conteneurisés via Docker.* »

Fournir le bon contexte, c'est permettre une réponse plus ciblée, plus pertinente, plus réaliste.

### 3. Intention : dire pourquoi vous posez la question

La qualité de l'échange dépend de la clarté du but visé.

*« Je veux que même un stagiaire puisse exécuter ce script sans risque d'erreur. »*

Nommer l'intention, c'est guider la forme, le ton, et le niveau de complexité de la réponse.

## 🗣 Le prompt est une conversation amorcée

Il est utile de voir le prompt non comme une requête, mais comme la **première phrase d'un échange**. Un bon prompt **ouvre l'espace de dialogue**, il invite à l'itération, à la reformulation, au rebond. Il pose un cadre... mais laisse de la place à la co-construction.

## Typologie des formes de prompts

Voici quelques formats fréquents que vous retrouverez dans la bibliothèque de motifs (chapitre 4) :

Type de prompt	Exemple	Usage typique
Contexte + Tâche	« Dans le cadre d'un service d'authentification OAuth2 en Go, écris un middleware... »	Implémentation ciblée
Exemple + Variation	« Voici une fonction JS. Peux-tu proposer une version plus performante avec <code>reduce</code> ? »	Refactor, optimisation
Roleplay	« Agis comme un expert Django senior. Quelles étapes pour refactorer cette application ? »	Conseil spécialisé, expertise simulée
Pas-à-pas	« Explique étape par étape comment sécuriser une API contre les attaques CSRF. »	Pédagogie, onboarding, formation
Cascade	« Ajoute un système de trace des actions dans des logs spécifiques »	Implémentation ciblée, Refactor, optimisation

## ✓ Bonnes pratiques

- Formatez vos prompts avec des **puces, blocs de code ou titres** pour structurer la pensée.
- Ajoutez des **exemples** : ils guident le modèle et clarifient vos attentes.
- Soyez explicite sur :
  - le langage et la version utilisés ;
  - le style ou niveau attendu ;
  - les contraintes spécifiques (techniques, fonctionnelles, organisationnelles).

## ✗ Erreurs fréquentes à éviter

- Empiler plusieurs demandes dans un seul prompt.

- Employer des termes flous : “améliore”, “rends ça plus propre”... sans critère.
- Oublier de formuler l’objectif réel derrière la tâche demandée.

## Exemple comparatif

**Prompt faible :**

« *Fais-moi une API Node.* »

➡ Résultat : réponse générique, peu exploitable.

**Prompt amélioré :**

« *Je veux créer une API REST en Node.js avec Express. Elle doit gérer des utilisateurs stockés dans MongoDB. Je souhaite une architecture modulaire, sans ORM, avec séparation claire des responsabilités. Peux-tu proposer une structure de fichiers et le code de base ?* »

✓ Résultat : réponse structurée, contextualisée, directement exploitable.

## 🛠 Fiche-outil — Structure d'un bon prompt

Élément	Exemple
<b>Contexte</b>	« Je travaille sur une API FastAPI en Python déployée sur AWS Lambda... »
<b>Tâche claire</b>	« ...je veux une fonction qui valide un token JWT dans les headers HTTP. »
<b>Contraintes</b>	« Sans ORM, logs clairs en cas d’échec, Python 3.10. »
<b>Intention</b>	« Le but est que ce soit compréhensible pour un développeur junior. »
<b>Format attendu</b>	« Exemple commenté + tests unitaires. »

## 📝 En résumé

Un bon prompt, c'est :

- 📋 une demande claire,
- un contexte explicite,
- une intention formulée,
- 📦 un format de réponse attendu.

C'est la base de toute collaboration fructueuse avec un LLM.

« Ce n'est pas l'IA qui est floue. C'est souvent notre manière de lui parler. »

*ChatGPT*

## Chapitre 2 — La grammaire de l'intention : penser et formuler avec un LLM

Un LLM ne comprend pas. Il complète. Il n'infère pas un raisonnement vrai, mais une suite plausible. C'est à nous, humains, d'en faire un partenaire valable — en cadrant l'échange, en le structurant, en l'habitant.

Concevoir avec un LLM, ce n'est pas lui donner des ordres. C'est construire un dialogue. Et comme tout dialogue, il a ses règles implicites, ses codes, ses zones de friction.

Dans ce chapitre, nous proposons une **grammaire de l'interaction** : un ensemble de gestes, de réflexes, de postures qui rendent le dialogue avec un LLM productif. Ce n'est pas une syntaxe à apprendre par cœur, mais une façon de penser : **penser en interaction**.

### ✍ Le LLM comme partenaire naïf

Imaginez une séance de conception avec un collègue ultra-compétent mais :

- qui ignore votre contexte exact,
- qui a une mémoire partielle de l'échange,
- qui répond parfois avec brio, parfois à côté,
- et qui n'ose jamais dire « je ne sais pas ».

C'est cela, travailler avec un LLM. Il faut donc créer les conditions d'un échange utile : structurer, contextualiser, itérer.

Le LLM connaît tout, mais ne sait rien de vous. Il est rapide, mais oublie. Il est créatif, mais naïf. Il n'est pas fiable par défaut — il le devient par collaboration.

### 🎯 Les 5 gestes fondamentaux de la grammaire d'intention

#### 1. Cadrer (toujours recontextualiser)

Un LLM ne possède ni mémoire longue ni connaissance de votre projet. Vous devez réinjecter le **contexte fonctionnel, technique, métier** dans chaque interaction.

*« Je travaille sur une application bancaire en Java, mon objectif est de sécuriser les appels à l'API de transaction. »*

#### 2. Questionner (une chose à la fois)

Le LLM fonctionne mieux avec des demandes unitaires. Une seule intention par prompt. Si vous lui posez trois questions, il répondra à celle qu'il comprend le mieux... pas forcément la plus importante.

✓ « *Peux-tu décomposer cette tâche en étapes techniques ?* » ✗ « *Donne-moi du code + une doc + les cas limites.* »

### 3. Reformuler (valider et clarifier)

À chaque réponse du modèle, interrogez la cohérence. Reformulez ce que vous avez compris. Provoquez des justifications. Cela crée un dialogue itératif.

« *Si je comprends bien, tu proposes une architecture orientée services. Quels sont les points faibles de cette approche selon toi ?* »

### 4. Synthétiser (consolider les décisions)

Le LLM n'a pas de continuité implicite. Il ne garde pas en tête ce qui a été dit plus tôt, sauf si vous le reformulez. Résumez les décisions, les hypothèses, les orientations prises à chaque étape importante.

« *Résumons les contraintes du système que nous avons posées : performance, tolérance aux pannes, faible coût. Peux-tu revalider les choix d'architecture à l'aune de ces critères ?* »

### 5. Tester (mettre à l'épreuve la réponse)

Ne prenez pas la réponse du modèle pour une vérité. Demandez-lui d'envisager un contre-exemple, une limite, un cas extrême. Cela affine la solution... ou révèle ses failles.

« *Dans quel cas cette solution pourrait échouer ?* » « *Et si le graphe contient des cycles négatifs ?* »

## 🔍 Cas d'usage : reformuler pour penser mieux

Une équipe travaille sur un module de gestion de stock. Elle utilise un LLM pour choisir entre une architecture monolithique et des microservices. Le prompt initial — « *Quelle architecture choisir ?* » — génère une réponse générique.

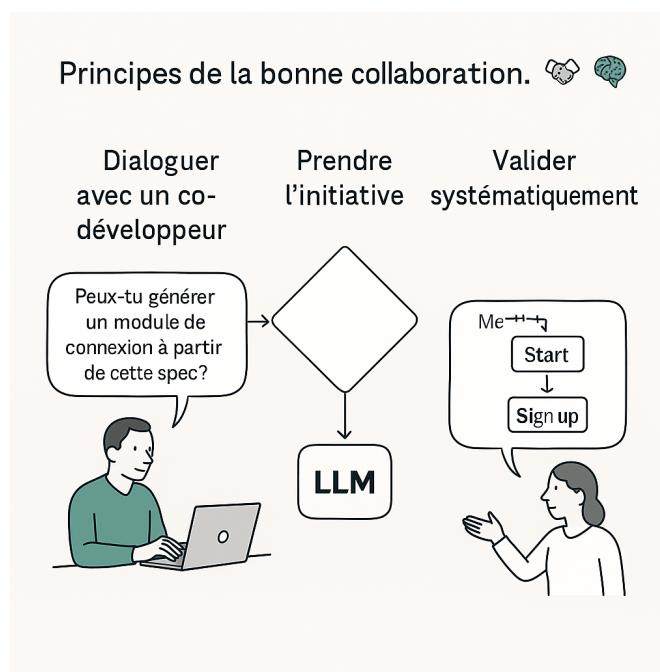
En injectant des contraintes spécifiques (taille de l'équipe, fréquence des déploiements, besoins d'évolutivité horizontale), la réponse s'affine. Le LLM devient alors un **simulateur d'options**, et le dialogue une façon d'explorer des possibles.

## ❖ Grammaire active — exemple d'atelier

Une autre équipe utilise un LLM comme **facilitateur d'idéation** lors d'un atelier. Chaque participant pose une question au modèle. La réponse est discutée collectivement, puis reformulée. Certains prompts deviennent des objets de travail communs. D'autres sont affinés en groupe. L'IA n'a pas remplacé la discussion : elle l'a catalysée.

### Synthèse : les 5 réflexes d'une bonne interaction

Geste	Question associée
Cadrer	Dans quel contexte suis-je ?
Questionner	Est-ce que je pose une seule question claire ?
Reformuler	Est-ce que je vérifie ce que le modèle a compris ?
Synthétiser	Est-ce que je stabilise ce qui a été décidé ?
Tester	Quelles limites n'ont pas été explorées ?



La grammaire de l'intention n'est pas une méthode figée. C'est un art d'interagir, d'ajuster, de construire du sens dans la nuance.

Comprendre cette grammaire, c'est poser les fondations d'un dialogue efficace. C'est apprendre à ne pas déléguer le raisonnement, mais à le distribuer. C'est, en somme, faire du LLM un **copilote intelligent**, et non un oracle à suivre aveuglément.

# Chapitre 3 — Les motifs du dialogue : construire un langage de conception avec les LLM

Concevoir avec un LLM, c'est plus qu'écrire des prompts. C'est pratiquer un art du dialogue. Ce chapitre propose une bibliothèque de **motifs conversationnels** — des séquences typiques d'interaction, issues du terrain, à la fois réutilisables et adaptables.

## Pourquoi un langage de motifs ?

Dans le développement logiciel, certaines situations reviennent sans cesse : formuler un besoin flou, explorer des options techniques, comprendre un code hérité, choisir une architecture. Avec un LLM, ces situations prennent une nouvelle forme — mais les **récurrences d'usage demeurent**.

Un motif, c'est une **forme récurrente d'interaction efficace** dans un contexte donné. Il ne dicte pas quoi faire, mais propose un **cadre pour bien faire**.

## 🔧 Structure d'un motif

Chaque motif suit une structure claire :

- **Nom** : une expression mémorable
- **Contexte** : quand le motif s'active
- **Problème** : ce qui empêche le progrès
- **Solution** : la posture ou forme d'interaction recommandée
- **Conséquences** : ce que cela permet
- **Exemple** : cas réel ou inspiré du terrain
- **Variantes (facultatif)** : déclinaisons utiles
- **Outils associés (facultatif)** : IDE, plugin, canevas...

## Bibliothèque de motifs

### ◆ Motif 1 — **Question Socratique** : reformuler pour comprendre

**Contexte** Un besoin est exprimé de manière floue ou partielle, ou vous entrez dans un domaine que vous maîtrisez peu.

**Problème** Un prompt vague produit une réponse générique ou mal orientée.

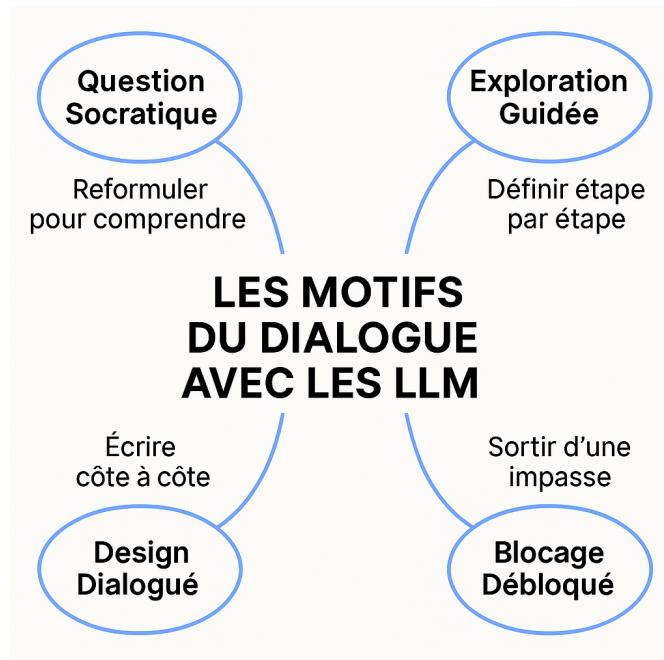
**Solution** Amorcer un **dialogue par questions progressives** pour clarifier l'intention, comme un facilitateur ou un coach :

« Quels types d'erreurs souhaitez-vous capturer ? » « Quel canal de notification ? Quelle fréquence ? »

#### Conséquences

- Clarifie les besoins, même pour l'humain
- Enrichit le prompt au fil du dialogue
- Engage un raisonnement partagé

**Exemple** Demander « Crée un script d'alerte » → réponse floue. Poser 3 questions ciblées → réponse précise, intégrée au projet réel.



## ◆ Motif 2 — Exploration guidée : découper pour mieux avancer

**Contexte** Vous devez explorer un domaine complexe ou inconnu.

**Problème** Les réponses sont trop généralistes. Trop de pistes, pas de structure.

**Solution** Demander au LLM de proposer **une décomposition en étapes**.

« Quelles grandes étapes pour sécuriser une API REST ? » « Propose-moi un plan d'implémentation progressif. »

### Conséquences

- Réduction de la charge cognitive
- Planification par itération
- Apprentissage ciblé sur chaque sous-partie

**Exemple** Migration monolithe → microservices : demander les étapes, puis explorer chaque étape une à une.

### Une session de pairing augmentée

Lors d'une séance de co-développement, deux développeurs travaillent ensemble à concevoir un module de traitement de factures. L'un d'eux propose d'interroger le LLM pour structurer le travail. Le prompt initial : *"Comment concevoir un module de traitement de factures dans un ERP ?"* produit une réponse dense mais confuse.

L'un des développeurs reformule alors : *"Peux-tu me proposer une décomposition en étapes pour construire ce module, du point de vue fonctionnel et technique ?"*

Le LLM répond :

- Identifier les sources de données (factures fournisseurs, clients).
- Définir les règles de validation.
- Modéliser les statuts de traitement.
- Intégrer les notifications.
- Prévoir l'export comptable.

À partir de cette réponse, les deux développeurs réorganisent leur backlog, définissent les premières user stories, et rédigent ensemble les spécifications du MVP. Le LLM a non seulement servi de facilitateur technique, mais aussi de médiateur de compréhension mutuelle, révélant des angles morts et clarifiant les priorités. L'exploration guidée a permis de sortir du flou initial pour entrer dans l'action concrète.

## ◆ Motif 3 — *Spécification inversée* : remonter aux intentions à partir du code

**Contexte** Vous devez comprendre ou refactorer un code sans doc, sans contexte.

**Problème** Le code précède la conception — impossible d'en déduire les intentions.

**Solution** Demander au LLM de **reconstituer la logique métier, les hypothèses, ou les user stories** à partir du code.

« Quelles règles métier vois-tu dans ce code ? » « Quelle user story ce bloc implémente-t-il ? »

### Conséquences

- Documentation rétroactive
- Détection des biais ou trous logiques
- Meilleure maintenabilité

**Exemple** Un script Python d'analyse réseau est soumis au LLM. Il reconstruit les intentions et génère les cas de test.

### Variantes

- 3.1 : *Reconstruction d'User Story*
- 3.2 : *Déduction des hypothèses implicites*
- 3.3 : *Contrat d'interface implicite*

#### Un LLM comme auditeur de code

Lors d'un audit technique sur un système de facturation, une équipe se retrouve face à un module PHP de plus de 800 lignes, écrit il y a 8 ans, sans tests ni documentation. Plutôt que de l'analyser ligne par ligne, l'équipe décide de le soumettre au LLM par blocs successifs.

À chaque itération, le prompt est : "Quels traitements réalise ce bloc de code ? Quelle règle métier cela semble-t-il implémenter ?"

Le LLM identifie la détection de doublons, la vérification de TVA, la gestion d'arrondis, et les cas particuliers de facturation croisée. Ces éléments sont mis en parallèle avec les processus métier décrits dans la documentation client, révélant plusieurs écarts non documentés.

Cette approche de spécification inversée a permis de :

- Documenter rétroactivement un système critique,
- Réconcilier la logique métier et l'implémentation réelle,

- Planifier une refonte progressive sans repartir de zéro.

## ⌚ Variantes du motif « Spécification Inversée »

### ✖️ Variante 3.1 : *Reconstruction d'User Stories*

Au lieu de demander uniquement *ce que fait le code*, on pousse le LLM à reformuler les intentions en *termes fonctionnels utilisateur*. Exemple de prompt :

« *En supposant que ce code corresponde à une fonctionnalité d'un produit, quelle user story pourrait-on en déduire ?* »

**Usage** : utile dans des projets où le code a été produit avant la formalisation des besoins (souvent le cas dans des prototypes ou des phases de hackathon).

### ✖️ Variante 3.2 : *Déduction des Hypothèses*

Demandez au LLM :

« *Quelles hypothèses implicites ce code semble-t-il faire sur les données, les contextes d'exécution ou les droits d'accès ?* »

**Usage** : précieux pour détecter des biais implicites, des présupposés sur les inputs, ou des angles morts en sécurité.

### ✖️ Variante 3.3 : *Contrat implicite*

Demandez :

« *Peux-tu expliciter un contrat d'interface pour cette fonction / ce module (types d'entrées, sorties, erreurs gérées) ?* »

**Usage** : aide à produire des *Design by Contract* à posteriori, ou à documenter des API sans doc initiale.

## ❖ Mises en pratique concrètes

### ✓ Pratique 1 : Générer les tests à partir de l'implémentation

Prompt-type :

« *Génère une suite de tests unitaires couvrant les cas normaux, limites et erreurs de cette fonction.* »

**Résultat** : Le LLM suggère souvent des cas non couverts dans le code, révélant des lacunes potentielles.

## ✓ Pratique 2 : Écrire la documentation avant refactorisation

Demande :

« Rédige une documentation technique synthétique (fonction, paramètres, exceptions, effets secondaires) à partir de ce code. »

**Usage** : crée un point d'ancrage avant transformation, utile pour la relecture et le pair programming.

## ✓ Pratique 3 : Déetecter les effets de bord

Prompt :

« Est-ce que ce code a des effets secondaires non visibles immédiatement (ex. : écritures sur disque, accès à des variables globales, dépendances réseau) ? »

**Usage** : audit de code legacy ou critique, base pour migration vers du code pur / testable.

## ⌚ Intégration dans un workflow d'équipe

Voici une **routine de revue de code assistée** utilisant ce motif :

1. 🔎 Sélectionner une fonction critique à revoir.
2. Demander au LLM de reformuler son intention métier.
3. ✓ Générer les tests que cette fonction est supposée passer.
4. ⚡ Comparer avec les tests existants.
5. 📝 Documenter ou enrichir à la volée.
6. ✖ Décider d'un refactor ou non.

⌚ **Outil compagnon possible** : Un plugin ou script intégré dans l'IDE qui, à la sélection d'une fonction, envoie automatiquement un prompt de spécification inversée au LLM et affiche les hypothèses métier dans une fenêtre latérale.

## Posture recommandée

- Ne pas se contenter de ce que le LLM dit, mais comparer avec les hypothèses de l'équipe.
- Utiliser les réponses comme **base de discussion**, notamment avec les parties prenantes non techniques.
- Croiser les approches : spécification inversée → exploration guidée → design dialogué.

## ◆ Motif 4 — **Modèle Miroir** : comparer pour éclairer un choix

**Contexte** Vous hésitez entre plusieurs implémentations possibles (par exemple, deux structures d'API, deux algorithmes, deux stratégies d'architecture), ou vous avez générée plusieurs variantes avec le LLM et souhaitez les évaluer de façon argumentée.

**Problème** Lorsqu'on explore des options seul ou en équipe, il est facile de se focaliser sur une solution "plausible" sans bien comprendre les différences, les conséquences ou les alternatives. Le LLM, sans guidance, tend à générer une seule réponse par défaut.

**Solution** Exploitez le LLM comme un **miroir comparatif**. Demandez-lui explicitement de produire *plusieurs* versions d'une même solution, puis de **les comparer lui-même selon des critères définis**. C'est une mise en tension productive entre options, qui stimule l'analyse critique.

« Propose trois structures possibles pour ce composant. Compare-les sur lisibilité, performance et maintenabilité. »

### Conséquences

- Analyse dialectique
- Explicitation des critères de choix
- Réduction du biais de confirmation

### Exemple concret

Lors d'un projet de refonte de système de paiement, l'équipe hésite entre :

1. Une architecture orientée événements avec Kafka.
2. Une architecture RESTful classique avec appels synchrones.

Le LLM est sollicité via ce prompt :

« Voici deux options d'architecture. Peux-tu détailler les avantages, risques et implications de chacune pour un système à haute disponibilité devant traiter 100 transactions par seconde ? »

Le modèle identifie que :

- Kafka apporte résilience et découplage, mais complexifie la supervision.
- REST est plus simple à auditer mais moins robuste en cas de pics de charge ou d'erreurs réseau.

Cette analyse partagée permet à l'équipe de trancher plus sereinement, *en conscience*.

Le LLM comme miroir d'équipe

Dans une réunion de design technique, une équipe ne parvient pas à se mettre d'accord entre deux styles de validation de formulaire côté frontend : impératif (en JS pur) ou déclaratif (via une lib type Formik ou React Hook Form).

Un développeur propose de demander au LLM une comparaison structurée.

Le prompt devient : « *Compare les styles impératif et déclaratif de validation de formulaire côté React. Donne des exemples de code et compare sur maintenabilité, UX et facilité de test.* »

Le LLM répond avec clarté, exemples à l'appui, révélant que le choix dépend du niveau de complexité métier attendu et de l'organisation du code. Cela permet à l'équipe de ne pas trancher "par opinion", mais sur des critères explicites.

Le modèle a servi ici de **facilitateur de clarification technique collective**, sans remplacer la décision humaine.

## ⌚ Variantes du motif « Modèle Miroir »

- **Miroir de styles** : comparer plusieurs styles de code pour une même logique (ex. fonctionnelle vs orientée objet).
- **Miroir de paradigmes** : comparer des approches (ex. polling vs event-driven).
- **Miroir de technologies** : comparer frameworks, langages, ou bibliothèques en fonction d'un usage ciblé.

## Motif 5 : *Clarification par contre-exemple* : Explorer les limites d'une proposition

**Contexte** : Après avoir reçu une réponse satisfaisante d'un LLM, il arrive que des doutes subsistent : le code proposé est-il robuste ? La solution envisagée tient-elle dans tous les cas ? Les limites implicites du raisonnement sont-elles comprises ?

**Problème** : Un prompt initialement bien formulé peut conduire à une réponse correcte mais trop optimiste ou générique. Le LLM tend à « deviner » une solution type, sans toujours prendre en compte les cas limites, les exceptions ou les erreurs de conception possibles. Cela donne l'illusion de maîtrise, là où il faudrait de la rigueur.

**Solution** : Demander explicitement un contre-exemple ou une situation qui ferait échouer la solution proposée. Cette approche vise à forcer la mise en lumière de zones grises, d'angles morts ou de présupposés implicites. On peut formuler par exemple :

« Quelle situation rend cette solution inefficace ? » « Et si les données sont mal formées ? »

Cette manière de questionner pousse le modèle (et le praticien) à réfléchir en creux, par la négation ou l'invalidation.

### Conséquences :

- Renforce la robustesse de la solution en mettant à l'épreuve ses fondements.
- Favorise une posture critique et antifragile dans la conception.
- Transforme le LLM en simulateur d'obstacles potentiels, utile pour la revue de code ou la documentation des limites d'un système.

**Exemple** : Dans un atelier sur la génération d'algorithmes de parcours de graphe, un étudiant a demandé à ChatGPT de lui fournir une version optimisée de Dijkstra en JavaScript. Le code généré semblait parfaitement fonctionnel. En testant ensuite la robustesse du résultat via la question : "Et si le graphe est orienté avec des cycles négatifs ?", le LLM a admis que l'algorithme proposé ne convenait pas, et a suggéré de basculer sur Bellman-Ford avec un test d'intégrité des poids. Cette interaction a permis non seulement d'enseigner les limites d'un algorithme, mais aussi d'aiguiser l'esprit critique face à la "bonne réponse".

### ➤ Posture réflexive : oser douter du bon élève

L'une des illusions les plus tenaces dans l'usage des LLM est celle de la "réponse parfaite" dès la première itération. Un motif comme "Clarification par contre-exemple" invite à adopter une posture scientifique : tester, falsifier, chercher ce qui ne va pas, même quand tout semble aller bien. Cela s'apparente à une relecture interne du raisonnement — une forme de revue de code dialoguée — où le développeur devient enquêteur des failles

possibles. C'est aussi une manière de former les plus jeunes à ne pas confondre autorité de l'outil et vérité absolue.

## ◆ Motif 6 — *Prompt piloté par les tests* : définir les attentes avant d'écrire

**Contexte** Vous créez un prompt pour un usage récurrent, mais les réponses sont fluctuantes.

**Problème** Sans attentes explicites, le LLM improvise. Les résultats sont inconsistants.

**Solution** Adoptez une démarche inspirée du Test-Driven Development : avant de rédiger le prompt, écrivez un ou plusieurs **tests d'intention** qui décrivent ce que vous attendez du LLM. Ce peuvent être :

- des exemples de sortie ;
- des critères formels (format, style, structure) ;
- des contraintes de contexte ou de contenu.

Utilisez ensuite ces tests pour guider l'écriture du prompt. Faites-le évoluer jusqu'à ce qu'il satisfasse les attentes définies.

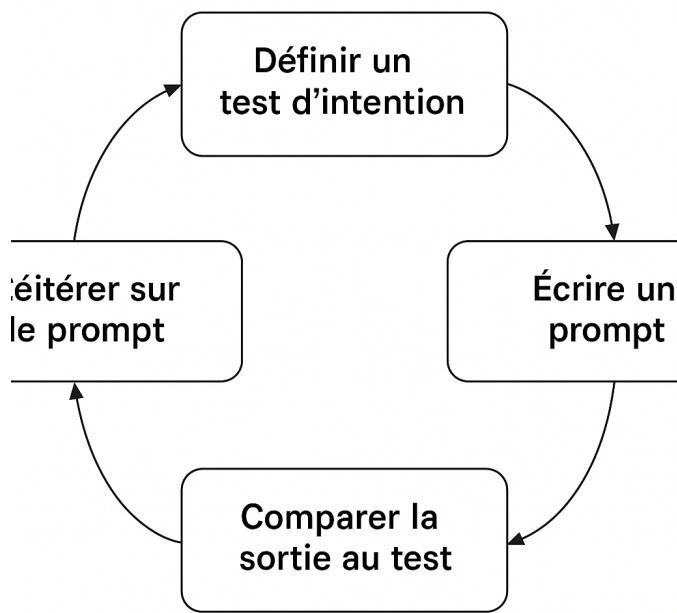
### Conséquences

- Vous gagnez en clarté sur ce que vous attendez vraiment du modèle.
- Vos prompts deviennent plus robustes, plus réutilisables et plus faciles à transmettre.
- Vous pouvez constituer une bibliothèque de prompts testés, versionnés et documentés.
- Le dialogue avec le LLM devient une activité d'ingénierie à part entière, structurée et maîtrisée.

**Exemple** Une équipe travaillant sur un assistant de support client voulait générer automatiquement des réponses types à partir de tickets. Le prompt initial donnait des textes trop longs, trop formels et parfois hors sujet. En définissant des tests d'attente clairs — *"Réponse ≤ 3 phrases, ton empathique mais professionnel, ne jamais inclure d'excuses juridiques"*, etc. —, l'équipe a pu concevoir un prompt beaucoup plus précis. Celui-ci a été intégré dans leur base de connaissance, avec des versions pour différents registres de langue selon le client.

### Variations

- **TDP visuel** : utilisez des "mock outputs" (sorties fictives attendues) comme référence.
- **TDP collaboratif** : faites définir les attentes par plusieurs membres de l'équipe (produit, technique, UX).
- **TDP dans le prompt** : incluez directement le test dans le prompt lui-même (ex. : *"Voici un exemple de réponse attendue..."*).



## 🎯 Motif 7 — *Reformulation Visuelle* : clarifier par la représentation

### Contexte

Vous utilisez un LLM pour concevoir un système, une fonctionnalité ou un processus. Le modèle vous fournit une solution textuelle relativement structurée (architecture, algorithme, flux d'information, découpage de responsabilités). Mais l'ambiguïté ou la complexité du texte rendent difficile une validation ou une mise en œuvre immédiate. Vous soupçonnez que des incohérences ou des zones d'ombre subsistent.

### Problème

Le langage naturel (ou pseudo-code) peut masquer des raccourcis logiques, des oubliés d'interfaces, des incompatibilités ou des malentendus. Les solutions proposées par le LLM semblent correctes, mais manquent parfois de précision dans la structure ou les interactions.

### Solution

Transformez la réponse du LLM en **schéma explicite**, puis reformulez ce schéma en **langage naturel structuré**, que vous soumettez de nouveau au modèle. Ce processus permet de :

1. Déetecter les zones ambiguës ou incohérentes,
2. Valider l'adéquation du modèle à l'intention initiale,
3. Enrichir ou compléter la proposition grâce au dialogue.

Le schéma peut être un diagramme de composants, de classes, de séquence, un cas d'usage, une carte mentale, ou un simple tableau structuré.

### Structure du dialogue

1. Prompt initial → Réponse LLM (textuelle)
2. Schéma du développeur (hors modèle)
3. Reformulation du schéma en langage naturel
4. Prompt de validation ou d'extension au LLM
5. Réponse LLM (corrections, suggestions, alternatives)

### Conséquences

- Permet de faire émerger des incohérences logiques plus tôt.
- Clarifie les échanges au sein de l'équipe (support visuel commun).
- Favorise l'appropriation de la solution par les humains.
- Renforce la capacité du LLM à produire des propositions robustes (en l'exposant à des feedbacks explicites).
- Développe des compétences de modélisation chez les développeurs.

### Exemple

Dans un projet de refonte d'un système de notifications multi-canaux, le LLM propose une architecture textuelle décrivant un service de gestion d'alerte, un module de priorisation, une file d'attente et un mécanisme de diffusion. Un développeur modélise cette proposition dans un diagramme de composants avec draw.io, puis reformule :

"J'ai interprété ta proposition ainsi : les alertes arrivent dans un gestionnaire, qui les classe, les stocke si besoin, et les transmet par webhook ou email. Redis sert de cache entre les modules. Ce schéma est-il cohérent ? Y a-t-il des points à améliorer ?"

Le LLM identifie une faiblesse : l'absence de gestion des échecs d'envoi. Il propose un mécanisme de retry avec journalisation, enrichissant la solution.

## Variantes

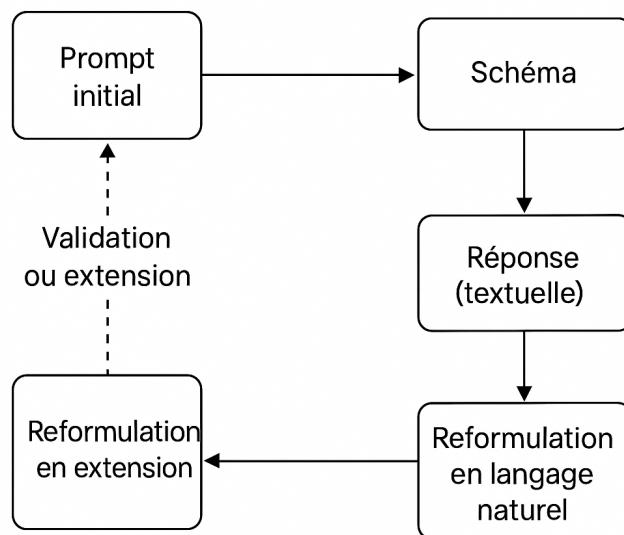
- Reformulation par **tableau à double entrée** (utile en cas de rôles et responsabilités).
- Utilisation d'**outils de modélisation UML**.
- Croisement avec des **cas d'usage rédigés** (user stories).
- Passage par **schéma manuscrit + transcription**.

## Anti-pattern

Ne pas reformuler visuellement des propositions complexes peut conduire à :

- une adhésion aveugle à des solutions incomplètes,
- une surconfiance dans les capacités du LLM,
- des malentendus entre co-concepteurs humains.

## Reformulation Visuelle



## ◆ Motif 8 — **Soin systémique** : investiguer les causes profondes d'un problème

**Contexte** Un comportement inattendu, un problème récurrent ou une tension d'équipe émerge sans cause évidente. La solution technique semble insuffisante ou incomplète. L'équipe cherche à comprendre "ce qui se joue vraiment" pour éviter de traiter seulement les symptômes.

**Problème** Les LLM sont souvent utilisés pour résoudre un problème exprimé, sans remettre en question la formulation du problème lui-même. Cela peut conduire à des solutions locales, sans impact durable, ou à des réponses rapides qui contournent les vraies causes. L'intention initiale est rarement interrogée en profondeur.

**Solution** Utiliser le LLM comme **partenaire d'investigation systémique**. Mobiliser un cadre comme le Neuf Pourquoi (Nine Whys) ou autre méthode d'analyse causale, en formulant des prompts qui explorent les hypothèses implicites, les causes possibles, les interactions systémiques. Demander ensuite au modèle de proposer **des pistes d'action ciblées** en lien avec les causes identifiées.

### Les Neuf Pourquoi : creuser le sens pour mieux agir

Inspiré des **Liberating Structures**, le canevas des *Nine Whys* propose un rituel simple, mais puissant : poser neuf fois de suite la question « **Pourquoi est-ce important pour toi ?** » à partir d'un sujet donné.

Loin d'être un interrogatoire, c'est un **chemin de clarification progressive**, où chaque réponse devient la base de la question suivante. On ne cherche pas une cause unique, mais une **profondeur de sens** : ce qui motive vraiment l'action, ce qui fonde les choix, ce qui compte profondément.

Dans le cadre du développement logiciel, cet outil devient précieux quand :

- une décision semble évidente mais suscite du flou ou de la résistance,
- un problème technique récurrent cache des tensions humaines ou systémiques,
- une équipe veut aligner ses efforts sur ce qui a du sens.

☞ Le LLM peut ici jouer un rôle de **facilitateur de questionnement** : en proposant des formulations de relance, en structurant les réponses, ou en révélant des contradictions implicites.

#### Exemples de prompts :

« *Peux-tu m'aider à simuler une session de Nine Whys sur ce problème : [décrire la situation] ?* » « *À chaque réponse,*

*propose une reformulation de "Pourquoi est-ce important ?" en changeant légèrement l'angle (valeurs, impact, émotion, système...). »*

En sollicitant un LLM avec ce cadre, on transforme une simple séance d'analyse en un **exercice de lucidité collective**, capable de reconnecter la technique au sens.

### Prompt-type

« Voici un problème récurrent dans notre équipe : [décrire la situation]. Peux-tu m'aider à explorer les raisons profondes en utilisant la méthode des 5 ou 9 Pourquoi ? »

« Pour chaque cause possible, quelles actions concrètes pourrions-nous envisager ? »

« Quelles causes sont techniques, relationnelles, organisationnelles ? »

### Conséquences

- Favorise une compréhension plus fine des tensions et blocages.
- Permet d'identifier des leviers d'action non évidents.
- Soutient une posture réflexive, systémique, non réactive.
- Offre un espace de délibération partagée avec l'IA comme catalyseur de recul.

**Exemple** Une équipe constate une perte de motivation chez les développeurs autour d'un module complexe. Le LLM est d'abord interrogé sur comment "remotiver" — mais les réponses sont superficielles. Le Scrum Master reformule avec :

*« Pourquoi cette démotivation selon toi ? Peux-tu explorer plusieurs causes possibles en t'appuyant sur les interactions humaines, les choix d'architecture, le rythme de livraison ? »*

Le modèle identifie alors :

- un manque de clarté dans les critères de qualité,
- une dette technique anxiogène,
- une absence de célébration des petits succès.

À partir de là, l'équipe définit plusieurs actions ciblées : des temps d'alignement sur les critères, des rituels de reconnaissance, un refactoring progressif. La cause systémique devient alors un levier d'action partagé.

### Variantes

- **8.1 — Arbre des causes** : demander au LLM de générer un arbre logique des causes

et sous-causes.

- **8.2 — Croisement de points de vue** : demander au modèle de formuler les causes vues par différents rôles (PO, dev, ops, client...).
- **8.3 — Hypothèses multiples** : inciter le modèle à produire des hypothèses contradictoires pour enrichir la réflexion.

## En résumé : la carte des motifs

Motif	Utilité principale
<b>Question socratique</b>	Clarification, cadrage
<b>Exploration guidée</b>	Décomposition, apprentissage
<b>Spécification inversée</b>	Compréhension, documentation
<b>Modèle miroir</b>	Choix argumenté, design
<b>Contre-exemple</b>	Robustesse, critique
<b>Prompt piloté par les tests</b>	Prompt stable et reproductible
<b>Reformulation visuelle</b>	Langage commun, validation collective
<b>Soin systémique</b>	Analyse causale, investigation en profondeur

Chaque motif est une brique. Ensemble, ils forment un langage. Ce langage n'est pas figé — il est fait pour évoluer avec vous.

# 👤 Chapitre 4 — Nouveaux rôles, nouvelles compétences : l'évolution des équipes augmentées

Concevoir avec un LLM, ce n'est pas seulement une question d'outillage. C'est un changement profond dans la manière dont les rôles s'exercent, se redéfinissent, se recomposent.

## ✖ Pourquoi ce chapitre ?

Nous avons vu comment interagir efficacement avec un LLM (chap. 2-3) et comment structurer ces interactions sous forme de motifs (chap. 4). Il est temps maintenant de comprendre ce que cela change, concrètement, dans la dynamique des équipes.

- Quels **nouveaux rôles** apparaissent ?
- Quelles **compétences deviennent critiques** ?
- Comment les **rôles existants se transforment-ils** ?

## Deux grilles de lecture

**1. Les rôles fonctionnels : ce que les gens font dans l'équipe (dev, PO, coach...)**

**2. Les postures conversationnelles : la manière dont on interagit avec le LLM (explorer, cadrer, vérifier...)**

Ces deux dimensions se croisent. Un développeur peut prendre tour à tour les postures de **concepteur**, de **curateur**, de **relecteur** — selon la tâche, le moment, le contexte.

## ⟳ Transformation des rôles existants

Rôle	Ce qui change avec le LLM
Développeur	Passe du “producteur de code” à “concepteur de dialogue structuré”
Tech Lead	Devient architecte de conversation, garant du sens produit
PO / PM	Peut rédiger des backlogs augmentés, scénariser des prompts
UX / UI	Peut générer des maquettes commentées, explorer des variantes
Coach agile	Intègre l'IA dans les boucles d'apprentissage, les rituels
QA / testeur	Génère ou vérifie des jeux de tests à partir de specs
Formateur / mentor	Utilise le LLM comme outil d'apprentissage, simulateur de dialogue

**Le développeur augmenté : un chef d'orchestre du raisonnement**

Le développeur moderne ne code plus seulement avec ses seules connaissances, mais mobilise un répertoire d'interactions avec des modèles qui savent compléter, reformuler, proposer, synthétiser. Ce changement appelle une nouvelle posture :

- **Anticiper la clarté du prompt** comme une compétence en soi.
- **Savoir détecter les biais, trous logiques ou généralisations abusives** dans les réponses.
- **Devenir le garant de l'intelligibilité** du système pour les humains à venir (lui-même, l'équipe, les auditeurs, les utilisateurs).

### Le développeur-éditeur

Un développeur aguerri m'a récemment dit : « *Je me sens plus proche d'un éditeur que d'un rédacteur. L'IA propose, je choisis, je coupe, je reformule, je structure.* » Ce parallèle avec le travail éditorial révèle bien la nouvelle nature de la production logicielle : elle n'est plus linéaire, mais interactive, critique, narrative.

## Postures émergentes dans le dialogue avec le LLM

### ⌚ Le Concepteur de prompts

- Maîtrise les subtilités de formulation.
- Construit des canevas, des séquences, des tests d'intention.
- Optimise pour clarté, robustesse, transférabilité.

### 🔍 Le Curateur / filtre sémantique

- Lit, corrige, nettoie, valide les réponses du LLM.
- Distingue signal de bruit, détecte les hallucinations.
- Structure les bonnes idées pour les intégrer.

### L'Explorateur de possibles

- Demande des variantes, des scénarios alternatifs.
- Joue avec les contre-exemples, les miroirs, les tests de limite.
- Utilise le LLM pour “penser à côté” et élargir le champ.

### 📚 Le Documentaliste augmenté

- Génère des résumés, des guides, des onboarding packs.
- Organise la connaissance produite par le LLM.
- Capitalise et partage les prompts efficaces.

### Le Concepteur de motifs

- Observe les récurrences d'usage.
- Formalise des motifs pour les transmettre.
- Participe à la culture d'équipe augmentée.

## Le Soigneur holistique

- Cherche à comprendre les causes profondes d'un problème au-delà de ses symptômes.
- Utilise le LLM comme partenaire d'investigation systémique : Neuf Pourquoi (Nine Whys), arbres des causes, hypothèses multiples.
- Élabore des pistes d'action ciblées selon les causes racines, en tenant compte des interactions humaines, techniques et organisationnelles.
- Adopte une posture d'écoute, de questionnement lent, de mise en lien des signaux faibles.

## Compétences transverses à renforcer

Certains savoir-faire deviennent transversaux à tous les métiers techniques :

Compétence	Pourquoi elle devient critique avec un LLM
Formulation claire	Condition de toute interaction efficace
Sens critique technique	Pour éviter de suivre aveuglément les réponses
Curiosité exploratoire	Pour tirer parti des suggestions inattendues
Capacité à synthétiser	Pour consolider les décisions issues du dialogue
Éthique de l'usage	Pour encadrer les biais, les risques, la documentation
Transmission des pratiques	Pour faire vivre les motifs, les capitaliser et les partager

## Cas d'équipe : le binôme augmenté

Dans un projet de refonte d'application, un développeur junior et un développeur senior travaillent en binôme avec un LLM. Le junior joue le rôle de **rédacteur de prompt**, propose des idées. Le senior filtre, restructure, donne du contexte. Le LLM devient un **troisième partenaire** silencieux, parfois inspirant, parfois fragile. Ensemble, ils co-conçoivent des modules documentés, testés, discutés.

## Cas concret : une rétrospective augmentée

Dans une équipe de développement, le Scrum Master a proposé d'utiliser un LLM comme co-facilitateur de la rétrospective. Les membres de l'équipe formulaient les irritants ou les réussites du sprint, et le modèle proposait des regroupements, des axes de réflexion, voire des pistes d'action. Le rôle du Scrum Master a évolué : il ne centralisait plus les idées, mais orchestrait une interaction triangulaire entre les voix humaines et la synthèse IA. Cette expérimentation a révélé de nouvelles compétences nécessaires : relecture critique, adaptation en direct, design de flux de dialogue.

## Vers de nouveaux rôles ?

Certains rôles émergents apparaissent dans les organisations :

- **Prompt Designer**
- **IA Facilitator**
- **Architecte Cognitif**
- **Curateur de connaissances générées**

Ces titres sont encore marginaux. Mais les fonctions qu'ils recouvrent sont déjà là, réparties de manière implicite dans les équipes. Les formaliser, c'est **reconnaitre que les gestes du design ont changé**.

## 💡 En résumé

- Les LLM **ne remplacent pas** les rôles existants — ils les augmentent, les hybridisent, les redistribuent.
- **De nouvelles postures** apparaissent : curateur, formateur, explorateur, synthétiseur...
- Les équipes doivent **s'outiller pour identifier, valoriser, transmettre** ces nouvelles compétences.

Un bon dialogue avec un LLM, c'est comme un bon design logiciel : clair, modulaire, itératif... et profondément humain.

# Chapitre 5 — Cartographier les usages : typologie des situations et des rôles

Concevoir avec un LLM, ce n'est pas appliquer une méthode linéaire. C'est **naviguer dans un espace d'interactions possibles**, qui varie selon le contexte, l'intention, le niveau de maturité. Ce chapitre propose une **carte de ces usages**.

## Pourquoi ce chapitre ?

Nous avons exploré :

- des **motifs conversationnels** (Chapitre 3),
- les **postures et rôles** qui émergent (Chapitre 4).

Il est temps maintenant de **cartographier les situations** dans lesquelles ces éléments s'activent. L'objectif : aider chacun à **reconnaître où il se trouve** dans la pratique, et à choisir les motifs ou postures les plus adaptés.

## Typologie des situations

Nous proposons ici six situations-types, fréquentes dans le travail logiciel augmenté par les LLM.

Situation	Intention principale	Rôle/posture activé(e)	Motifs typiques
<b>Exploration</b>	Découvrir un domaine, une techno, une approche	Explorateur	Exploration guidée, Miroir
<b>Cadrage</b>	Clarifier un besoin flou ou implicite	Formulateur, facilitateur	Question socratique, Décomposition
<b>Refactorisation</b>	Améliorer un existant	Analyste, critiqueur	Spécification inversée, Contre-exemple
<b>Documentation</b>	Générer ou reconstruire du sens	Synthétiseur, documentaliste	Spécification inversée, Résumé ciblé
<b>Validation</b>	Vérifier une solution, tester un raisonnement	Curateur, relecteur	Prompt piloté par les tests, Contre-exemple
<b>Co-conception</b>	Créer à plusieurs avec un LLM comme partenaire	Facilitateur, co-concepteur	Miroir, Clarification, Synthèse

## Exemple 1 — Situation “Exploration”

**Contexte** : un développeur fullstack découvre le pattern CQRS.

**Posture** : explorateur, apprenant actif **Prompt** : « Explique-moi CQRS étape par étape, avec un exemple Node.js. » **Motifs activés** :

- *Exploration guidée* (pour la découverte)
- *Contre-exemple* (pour tester la compréhension)
- *Miroir* (CQRS vs CRUD)

## Exemple 2 — Situation “Cadrage flou”

**Contexte** : une équipe reçoit une demande métier mal formalisée.

**Posture** : facilitateur, analyste **Prompt** : « Voici les éléments métier reçus. Peux-tu aider à formaliser une user story complète avec critères d'acceptation ? » **Motifs activés** :

- *Question socratique*
- *Spécification inversée*
- *Reformulation par test*

## Exemple 3 — Situation “Refactorisation guidée”

**Contexte** : un dev reprend une fonction critique non testée ni commentée.

**Posture** : critiqueur, nettoyeur **Prompt** : « Que fait ce code ? Quelle serait une version plus claire, avec tests associés ? » **Motifs activés** :

- *Spécification inversée*
- *Contre-exemple*
- *Miroir de style*

## Vers une carte d'usage dynamique

On peut imaginer cette cartographie comme une **matrice vivante**, dans laquelle :

- Chaque **situation** active une combinaison de postures et de motifs.
- Ces combinaisons peuvent **évoluer avec l'expérience**.
- Certaines équipes documentent leurs propres cartes d'usage (quels motifs pour quelles tâches ?), pour **faciliter l'onboarding ou les revues**.

## Cas d'équipe : usages hybrides

Dans une startup, deux développeuses utilisent un LLM pour concevoir un microservice d'authentification. Elles alternent :

- **Exploration** de l'approche OAuth2
- **Co-conception** d'un middleware
- **Documentation** des choix
- **Validation** par génération de tests

Elles changent de posture selon l'étape. Le LLM devient une **surface partagée** de réflexion.

## ✏️ En résumé

- Les situations-types offrent un **repère pratique** pour mobiliser les bons motifs.
- Les rôles et postures ne sont pas figés : on **circule entre eux selon le moment**.
- Cartographier ses usages, c'est aussi **prendre conscience de sa maturité d'interaction**.

Ce que vous faites avec un LLM dépend moins de l'outil... que de votre intention, votre posture, et votre capacité à choisir le bon motif au bon moment.

## ❖ Chapitre 6 — Intégrer les motifs au quotidien : entre travail individuel et pratiques d'équipe

Un motif n'a de valeur que s'il est vécu, adapté, partagé. Ce chapitre propose des façons concrètes d'**ancrer les motifs dans votre quotidien**, que vous soyez en solo ou en équipe.

### ◆ Pratique individuelle : un dialogue au long cours

#### 💻 1. Travailler avec un LLM comme partenaire personnel

Un LLM peut devenir :

- un compagnon de réflexion,
- un simulateur d'options,
- un conseiller technique,
- un miroir de votre propre pensée.

Il ne remplace pas votre expertise. Il l'amplifie — si vous savez poser les bonnes questions.

#### 👉 2. Développer ses motifs personnels

En expérimentant, vous allez découvrir vos propres séquences efficaces. Par exemple :

- Une forme de prompt qui vous convient mieux,
- Une manière d'introduire un contexte projet,
- Une méthode pour obtenir une réponse exploitable.

Documentez-les. Donnez-leur un nom. Partagez-les.

Un bon motif est souvent **né d'une frustration** bien surmontée.

#### 📘 3. Capitaliser avec une “bibliothèque personnelle de prompts”

Gardez trace de vos meilleures interactions :

- Ce qui a marché,
- Ce qui a échoué,
- Ce que vous avez reformulé.

Utilisez une note, un outil de gestion du savoir (ex. Obsidian, Notion), ou un simple fichier Markdown. Cette base devient un **référentiel d'apprentissage et de transmission**.

### Pratiques d'équipe : un langage commun qui se

# construit

## 👤 1. Partager les motifs vécus

Lors d'un stand-up, d'une revue, ou d'un débrief :

- « Quel motif as-tu utilisé ? »
- « Ce prompt, tu l'as construit comment ? »
- « Quels tests d'intention as-tu posés ? »

Ces questions rendent **visible** la pratique conversationnelle avec l'IA.

## 2. Revue augmentée par motifs

En code review, ajoutez une dimension “dialogue avec le LLM” :

- Montrer les prompts utilisés,
- Expliquer les arbitrages faits avec l'IA,
- Questionner la fiabilité ou la logique de certaines suggestions.

Ce n'est plus seulement le code qu'on révise : c'est **le chemin de pensée** qui y mène.

## ❖ 3. Design dialogué en binôme

Un binôme travaille avec un LLM : 1 pose les questions, 2 observe, reformule, propose une variation. Le prompt devient un **objet partagé, construit à deux, testé ensemble**.

Ce rituel peut faire émerger des **motifs d'équipe** : → “Voici notre manière d'explorer un design.” → “Voici notre canevas de refactorisation.”

## 🛠 Ateliers possibles

Atelier	Objectif	Durée	Format
<b>Motifs en revue de code</b>	Identifier les motifs utilisés / à améliorer	30 min	Pair review
<b>Prompt kata</b>	Pratiquer un motif en situation guidée	45–60 min	Atelier en binômes
<b>Cartographie d'usage</b>	Identifier les motifs utilisés par l'équipe	45 min	Atelier mural ou Miro
<b>Retex de conversation</b>	Partager une interaction réussie ou ratée	15 min	Stand-up ou rétrospective
<b>Création de motifs d'équipe</b>	Formaliser un motif vécu par l'équipe	60 min	Atelier collaboratif

## 📚 Cas d'usage : l'équipe qui se dote de son propre

# langage

Une équipe agile décide de créer son “dialecte” d’interaction avec les LLM. À chaque sprint, elle :

- documente les prompts efficaces,
- formalise les motifs émergents,
- crée un “prompt book” commun.

Au fil du temps, ces éléments deviennent :

- des **supports de formation** pour les nouveaux,
- des **outils de discussion** en rétro ou en review,
- des **invariants** dans leur pratique de conception.

## L’atelier du vendredi

Dans une équipe toulousaine, chaque vendredi matin est consacré à un “Atelier IA”. Chaque membre partage une interaction marquante avec un LLM durant la semaine. Un tableau Kanban en ligne recense les motifs utilisés, les prompts testés et les résultats obtenus. Cela a permis à l’équipe non seulement d’améliorer la qualité de ses prompts, mais aussi de créer un référentiel commun vivant, nourri par les expériences de chacun.

## Créer un environnement favorable

Certains prérequis culturels et organisationnels facilitent grandement l’intégration des motifs :

- **Une culture de l’expérimentation** : permettre aux développeurs de tester des approches sans crainte de l’échec.
- **Une confiance dans l’autonomie** : laisser aux équipes la liberté de choisir quand et comment interagir avec les LLM.
- **Une reconnaissance du temps réflexif** : ne pas valoriser uniquement la production immédiate, mais aussi les temps de dialogue et de reformulation.

## Outils et supports

Pour faciliter l’usage des motifs dans les contextes professionnels, plusieurs outils peuvent être mobilisés :

- **Des bibliothèques de prompts contextualisés**, classés par types de tâche (refactoring, documentation, tests, modélisation, etc.).
- **Des canevas visuels** pour guider les dialogues complexes (ex. : arbres de décision pour affiner les architectures).
- **Des extensions d’IDE** intégrant les motifs les plus fréquents comme raccourcis ou assistants intégrés.

## 💡 En résumé

- Les motifs deviennent puissants **quand ils sont partagés et incarnés**.
- Travailler avec un LLM est une pratique vivante, **qui gagne à être racontée**.
- Le langage de motifs est un **outil d'alignement**, de réflexion, de transmission.

Une équipe augmentée, ce n'est pas une équipe qui utilise une IA. C'est une équipe qui **apprend à dialoguer avec elle, ensemble**.

# ॥ Chapitre 7 — Responsabilité, transparence et limites : une éthique du développement augmenté

Utiliser un LLM dans le développement, ce n'est pas seulement une opportunité. C'est aussi une responsabilité. Il ne suffit pas que le résultat fonctionne. Il faut **pouvoir expliquer comment il a été produit, et à quelles conditions.**

## Pourquoi ce chapitre ?

Dans un contexte où :

- des outils proposent du code sans auteur clair,
- des équipes intègrent des blocs générés sans les comprendre,
- des décisions d'architecture sont prises à l'aide de suggestions IA,

la **documentation des interactions avec les LLM** devient un enjeu majeur. Non pas pour tout consigner... mais pour **rendre visible ce qui a été généré, validé, interprété**.

## 📘 Partie 1 — Documenter l'usage des LLM

### 1.1 Pourquoi documenter ?

- Pour garder une trace des choix faits avec l'aide de l'IA.
- Pour éviter la **dette générative** : du code produit trop vite, sans explication.
- Pour pouvoir réexaminer un raisonnement ou un prompt dans six mois.
- Pour outiller les relecteurs et les équipes QA.

La documentation d'un prompt n'est pas un luxe. C'est **une condition de la maintenabilité**.

### 1.2 Que documenter ?

Élément	Objectif
<b>Prompt source</b>	Comprendre l'intention initiale
<b>Version du LLM utilisé</b>	Évaluer les limites, biais ou hallucinations potentielles
<b>Réponse générée</b>	Historiser l'itération utilisée
<b>Validation humaine apportée</b>	Identifier le rôle de l'humain dans l'acceptation du résultat
<b>Hypothèses contextuelles</b>	Préserver la logique derrière la génération

### 1.3 Formats possibles

- Annotation en commentaire dans le code

- Historique dans l'outil LLM (chat, snapshot, fichier .prompt.md)
- Documentation à part (Wiki, PR, fichier prompts/)
- Modèle structuré (ex. Fiche Prompt + Tests d'intention associés)

## 1.4 Exemple concret

```
// Fonction générée à partir d'un prompt GPT-4 le 12/04/2025
// Prompt : "Écris une fonction en JavaScript qui valide une adresse mail avec une RegExp simple"
// Réponse modifiée pour :
// - Ajouter la gestion des caractères spéciaux
// - Remplacer l'alerte par une exception explicite
```

# ॥ Partie 2 — Enjeux éthiques et responsabilité

## 2.1 LLM = responsabilité partagée

Ce n'est pas parce qu'un LLM a proposé un code que vous en êtes moins responsable. Vous êtes responsable **de ce que vous comprenez, validez, intégrez.**

Les modèles sont puissants, mais :

- ne donnent aucune garantie de fiabilité,
- peuvent reproduire des biais,
- peuvent générer du contenu non conforme ou juridiquement risqué,
- ne sont pas capables de refuser une tâche inappropriée par eux-mêmes.

### Un bug venu d'un exemple convaincant

Un développeur a récemment intégré un snippet de code généré par LLM pour l'authentification OAuth. Le code était syntaxiquement parfait, commenté, et semblait sécurisé... sauf qu'il utilisait une bibliothèque obsolète et vulnérable. L'audit de sécurité a révélé une faille critique. Le LLM avait simplement "recopié" un exemple daté, sans signaler de mise en garde. Résultat : plusieurs jours perdus, et une prise de conscience utile.

## 2.2 Risques fréquents

Risque	Exemple
<b>Hallucination de fonction</b>	Fonction plausible mais non existante dans un langage donné
<b>Copie involontaire</b>	Reproduction d'un bout de code protégé issu du corpus d'entraînement
<b>Biais implicite</b>	Stéréotypes dans les exemples ou réponses générées

<b>Surconfiance</b>	Prise de décision sans relecture ni test, sur la base d'un prompt unique
<b>Manque de traçabilité</b>	Code généré sans indication de son origine ni de sa validation

## 2.3 Questions à se poser (checklist éthique)

1. Ai-je compris ce que le modèle a produit ?
2. Puis-je expliquer à quelqu'un pourquoi cette solution est valable ?
3. Ai-je testé ou vérifié ce code ?
4. Ai-je signalé qu'il a été généré ?
5. Le modèle a-t-il produit une réponse biaisée ou discutable ?
6. Cette interaction pourrait-elle être mal interprétée ou mal réutilisée par quelqu'un d'autre ?
7. Est-ce que j'assumerais cette décision en production ?

Si la réponse est "non" à deux questions ou plus, il est **trop tôt pour valider cette contribution IA.**

## 🔍 Vers une culture de la transparence

- Rendre visible l'usage des LLM n'est pas une contrainte. C'est **un levier de confiance collective.**
- Cela permet de relire, de corriger, de transmettre.
- Cela constitue une **preuve de diligence technique** en cas de litige ou d'incident.
- Cela alimente une culture d'équipe où l'IA **stimule le raisonnement plutôt qu'elle ne le remplace.**

### Le "Journal du dialogue"

Dans une startup du secteur santé, chaque interaction avec un LLM pour des sujets critiques (protocoles, anonymisation, sécurité) est archivée sous forme de journal. Ce journal inclut : prompt initial, itérations, choix retenus, évaluation humaine, et justification des décisions. Ce dispositif améliore la transparence interne, facilite les audits, et cultive une posture réflexive.

## 🔒 Protéger les données, même dans le dialogue

*Tout ce que vous envoyez à un LLM n'est pas neutre — ni invisible.*

Les interactions avec un LLM peuvent exposer involontairement des données sensibles, confidentielles ou personnelles : noms de clients, extraits de code propriétaire, exemples de production, ou encore décisions stratégiques.

Même lorsque l'outil semble local ou « sécurisé », il est essentiel d'adopter une posture de prudence active :

- **Filtrer en amont** les données transmises, comme on le ferait pour une publication publique.
- **Éviter les copier-coller aveugles** issus de documents confidentiels ou de bases internes.
- **Utiliser des environnements contrôlés**, capables de garantir la non-exploitation des données (LLM auto-hébergés, mode entreprise, clauses contractuelles explicites).
- **Anonymiser les données** utilisées dans les prompts, dès que possible.
- **Former les équipes** aux risques liés à la fuite involontaire d'information via un prompt mal formulé.

Enfin, se poser une question simple avant chaque envoi :

*"Aurais-je le droit d'envoyer ceci par email à une tierce personne extérieure à mon organisation ?"* Si la réponse est non, alors le prompt doit être retravaillé.

Ce souci de **protection des données** s'inscrit dans une éthique plus large : celle d'un développement **responsable, traçable et conscient de ses impacts** — techniques, sociaux et légaux.

## ❸ En résumé

- La documentation des prompts et des interactions est une **bonne pratique technique** et un **geste éthique**.
- Les LLM déplacent la responsabilité, mais ne la dissolvent pas.
- Seule une **pratique transparente et partagée** peut garantir la qualité, la robustesse et l'éthique des conceptions assistées par IA.

Les LLM ne pensent pas. Ils complètent. Mais vous, vous **pensez avec eux** — et cela vous engage.

## ⌚ Chapitre 8 — Une agilité augmentée ?

Et si le LLM devenait un nouveau type de membre de l'équipe ? Non pas un développeur automatisé, mais un **stimulateur de conversation**, un **accélérateur de réflexion**, un **miroir de posture**.

### Pourquoi ce chapitre ?

L'agilité, dans son essence, repose sur des cycles courts, une adaptation constante, une collaboration étroite. Les LLM peuvent sembler, à première vue, extérieurs à cette culture humaine et incrémentale.

Et pourtant... bien utilisés, ils peuvent **accélérer certains flux**, enrichir la réflexion collective, ou faciliter l'appropriation de pratiques.

Ce chapitre explore **comment l'agilité peut être augmentée par les LLM**, sans perdre son âme.

### ➡ Partie 1 — Les principes agiles, revisités par l'IA

Principe agile	Ce que le LLM permet (ou interroge)
Collaboration constante	Simuler des points de vue, prototyper des idées, challenger un choix
Réponse au changement	Réviser rapidement des specs ou du code en réponse à une nouvelle contrainte
Simplicité	Reformuler une solution complexe pour en extraire l'essence
Feedback rapide	Générer des tests, comparer des variantes, explorer des alternatives
Travail en équipe auto-organisée	Outiliser la prise de décision, formaliser les idées émergentes

Le LLM ne remplace pas l'équipe. Il **outille sa réflexivité**.

## 🔧 Partie 2 — Rituel par rituel : que change le LLM ?

### 1. L'IA comme pair silencieux dans la planification

Durant les *sprint plannings*, un LLM peut être utilisé pour clarifier des user stories, en générer des variantes, ou estimer des scénarios alternatifs. Par exemple, une équipe produit peut demander :

*“Quels cas limites devrions-nous prendre en compte pour cette user story ?”*

ou encore :

*“Peux-tu proposer trois façons différentes d’implémenter cette fonctionnalité avec leurs avantages/inconvénients ?”*

Le modèle ne prend pas la décision, mais il élargit l'espace de réflexion.

## 2. Écriture de tests et de critères d'acceptation

L'un des usages les plus directs en agilité est la génération (ou la vérification) de **tests automatisés** ou de **critères d'acceptation** à partir d'une user story. Cela aligne naturellement les équipes sur le “*definition of done*”, tout en réduisant les oubli.

*User story : En tant qu'utilisateur connecté, je veux recevoir une notification quand un nouveau message arrive. → Critères d'acceptation générés :*

- L'utilisateur est connecté.
- Un message est reçu.
- Une notification apparaît dans les 3 secondes.
- L'utilisateur peut cliquer sur la notification pour ouvrir le message.

## 3. Daily meetings augmentés

Sans remplacer les échanges humains, un LLM peut aider à synthétiser les points clés discutés la veille, ou à générer un résumé quotidien des tickets en cours, des blocages identifiés, ou des dépendances critiques. Intégré dans un outil comme Jira ou GitHub, cela libère du temps pour des échanges plus qualitatifs pendant les dailies.

**Exemple :** une équipe a connecté un LLM à son tableau Kanban. Chaque matin, un résumé automatisé des mouvements sur le board était proposé. Cela a permis de gagner en réactivité sur les points bloquants.

### Actions rapides

- Reformuler les blocages en prompt pour les rendre actionnables
- Générer rapidement des solutions de contournement
- Partager les prompts utilisés la veille comme “retex instantané”

*“Hier j'ai utilisé un miroir d'implémentation pour débloquer ma PR.”*

## 4. L'IA comme miroir en rétrospective

L'un des usages émergents les plus intéressants est celui de l'**analyse réflexive assistée**. En analysant les logs de tickets, les commentaires de code ou les transcriptions de réunion, un LLM peut détecter des motifs récurrents de tension, de délai, ou de manque de clarté.

*"Sur les trois derniers sprints, quelles user stories ont nécessité plus de deux itérations de test ?" "Peux-tu repérer des points communs entre les bugs signalés en production ?"*

Cela n'exclut pas l'intelligence collective humaine, mais permet de **poser de meilleures questions** lors des rétrospectives.

## 5. Bonnes pratiques pour une intégration saine

- **Co-construire les usages** : l'équipe doit décider collectivement quand et comment utiliser les LLM.
- **Maintenir la transparence** : documenter les échanges avec le modèle, archiver les prompts clés.
- **Conserver le contrôle humain** : l'IA propose, mais l'équipe décide.
- **Favoriser l'apprentissage mutuel** : une veille régulière sur les usages IA en équipe peut créer une culture de progression continue.
- **Identifier les biais** : toujours valider les suggestions du LLM, notamment sur les choix d'architecture, de sécurité ou de performance.

**« L'IA est-elle un membre de l'équipe ? »**

C'est une question qui revient souvent. Un LLM peut-il être considéré comme un *membre virtuel* de l'équipe ? Pour certains, cela aide à le personnifier et à structurer les interactions (ex. : "notre assistant d'équipe"). Pour d'autres, cela dilue la responsabilité collective. Une position intermédiaire consiste à le voir comme **un outil de facilitation intelligente**, à la fois accessible à tous et gouverné par des règles d'usage partagées.

## 🚩 Partie 3 — Risques et vigilance

Risque	Explication / Exemples
<b>Surcharge cognitive</b>	Trop de prompts, trop de suggestions à trier
<b>Effet tunnel génératif</b>	Suivre une suggestion IA sans la remettre en question
<b>Illusion de vitesse</b>	Générer vite ≠ avancer mieux
<b>Biais dans les décisions</b>	Le modèle propose une "moyenne" sans tenir compte du contexte réel
<b>Automatisation aveugle</b>	Remplacer les conversations humaines par des réponses IA non discutées

L'agilité est une culture du feedback. Le LLM ne doit **jamais supprimer la boucle humaine.**

## ✎ En résumé

- L'agilité n'a pas besoin d'être remplacée. Elle peut être **augmentée** par un usage réfléchi des LLM.
- Les **rituels deviennent des lieux d'activation des motifs.**
- Le LLM devient un **outil de facilitation, pas un automate de production.**
- Cela suppose une **vigilance collective**, un cadre d'usage, et un langage partagé.

Un LLM bien utilisé rend l'équipe plus autonome, plus réflexive, plus alignée.  
Mal utilisé, il peut **court-circuiter les fondamentaux** de l'agilité.

# 8 Chapitre 10 — Cadres de mise en œuvre : ateliers, méthodes et rituels pour une pratique augmentée

Voici le terrain d'expérimentation : des formats pour apprendre ensemble, explorer, tester, documenter et transmettre les usages de l'IA dans vos équipes.

Après avoir exploré les motifs, les principes et les scénarios du développement augmenté, ce chapitre propose des **formats concrets** pour intégrer ces pratiques dans la réalité quotidienne des équipes. Ateliers, rituels, canevas, jeux sérieux : il s'agit de rendre tangibles les apports des LLM dans des dynamiques collectives, sécurisées et apprenantes.

## 1. Atelier “Design de prompt en équipe”

**Objectif :** Apprendre à formuler, reformuler et tester des prompts collectivement pour un cas de conception ou de développement réel.

- **Durée :** 1h30 à 2h
- **Participants :** 3 à 6 personnes (dev, PO, UX, test, coach...)
- **Matériel :** Accès à un LLM, paperboard ou miro/whiteboard/draft.io, canevas de prompt

**Déroulé :**

1. Choisir une problématique réelle (ex. : “Comment découper ce module en services ?”)
2. Écrire un premier prompt naïf ensemble
3. Identifier les manques, les ambiguïtés, les hypothèses implicites
4. Itérer, enrichir, comparer plusieurs formulations
5. Analyser les réponses générées : lesquelles éclairent la réflexion ?
6. Extraire ensemble un **patron de prompt réutilisable** pour l'équipe

**Résultat :** Des prompts testés + une meilleure capacité collective à dialoguer efficacement avec les IA

## 2. Rituel “Daily du dialogue”

**Objectif :** Installer un rituel court (5 à 10 min) où l'équipe partage un retour d'expérience sur une interaction LLM.

**Format léger, quotidien ou hebdo :**

- Qu'ai-je tenté avec l'IA ?
- Quelle surprise ? Quel biais ? Quelle bonne idée ?
- Ce que j'en retiens ou ce que je propose d'essayer

**Effet :** Culture réflexive, apprentissage collectif, veille sur les limites et les bonnes pratiques

### 3. Atelier “Cartographie des motifs de dialogue”

**Objectif :** Identifier les motifs d’interactions LLM les plus utilisés ou désirables dans l’équipe.

- **Inspiré du pattern language**
- **Durée :** 2h
- **Support :** cartes de motifs (types de dialogue), tableau à double entrée : *efficacité perçue vs fréquence d’usage*

#### Exemples de motifs :

- Reformulation d’une idée floue
- Génération de cas de tests
- Exploration d’alternatives d’architecture
- Traduction d’un besoin métier en user story
- Explication pas à pas d’un comportement

**Sortie possible :** Un “grimoire des dialogues utiles”, propre à l’équipe

### 4. Le jeu des prompts absurdes

**Objectif :** Expérimenter les limites, les paradoxes, les hallucinations — avec humour.

#### Règles :

- Chaque participant écrit un prompt volontairement absurde, contradictoire ou piégeux.
- Le LLM doit répondre sérieusement.
- On débrieve : pourquoi le modèle a-t-il suivi cette logique ? Que révèle cette erreur ?

**But :** Apprendre à repérer les zones à risque, à formuler de manière robuste, à relativiser les réponses IA

### 5. Référentiel d’équipe “LLM Ready”

**Un guide interne**, construit par et pour l’équipe, intégrant :

- Bonnes pratiques de prompting
- Modèles de prompts testés
- Risques connus (ex. : hallucinations en matière de sécurité, mauvais découpage métier...)
- Grille d’évaluation des réponses (pertinence, sécurité, robustesse, cohérence)

- Niveaux de criticité : quand valider avec un humain, quand automatiser ?

**Format vivant** : sous forme de Notion, wiki, Miro, ou README.

**But** : Rendre les usages LLM explicites, conscients, partagés et adaptables

## Une ingénierie augmentée est aussi une ingénierie sociale

Ces formats montrent que le développement augmenté ne se résume pas à l'outillage. Il repose sur :

- une culture du dialogue (avec l'IA et entre humains),
- une capacité à expliciter nos raisonnements,
- une pratique réflexive qui transforme l'équipe autant que les livrables.

# Chapitre 10 — Transmettre, former, partager les motifs

Un motif, par nature, est fait pour être partagé.

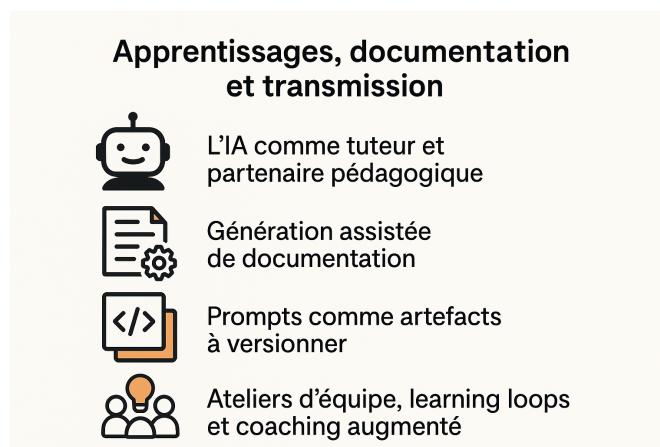
Ce n'est pas une astuce individuelle, mais un **savoir incarné, transmis, enrichi en contexte**.

## 🎯 Pourquoi ce chapitre ?

Un langage de motifs ne vaut que s'il est :

- **pratiqué,**
- **questionné,**
- **intégré,**
- **transmis.**

Ce chapitre répond à une question clé : **comment faire vivre les motifs dans votre organisation, votre communauté ou votre écosystème de pratique ?**



## 🏢 Formation initiale : apprendre par la pratique

### 1. Ateliers d'initiation

Atelier	Objectif	Durée	Format
<b>Prompt Dojo</b>	Créer un prompt, l'itérer, le tester	1h	Binômes / petits groupes
<b>Exploration par motifs</b>	Résoudre un problème en utilisant un motif	1h30	Cas d'usage réel
<b>Décryptage de conversation IA</b>	Analyser une interaction générée	45 min	Exercice en miroir

Chaque atelier est centré sur un ou deux **gestes conversationnels**. Il vise l'appropriation, pas l'exhaustivité.

# Transmission continue : faire vivre les motifs dans l'équipe

## 📘 2. Partage de motifs vécus

Exemples de pratiques :

- Réserver 10 min en rétrospective pour “le motif du sprint”.
- Tenir un **journal d'équipe** des interactions LLM remarquables.
- Ajouter un champ “motif utilisé” dans les PR.

Un motif partagé devient un **point d'appui pour aligner les pratiques**.

## 3. Crédit de motifs d'équipe

Une équipe peut créer ses propres motifs :

1. Choisir une situation fréquente (ex. : "refactorisation après bug").
2. Identifier ce qui fonctionne avec le LLM.
3. Nommer le motif, lui donner une forme.
4. Le documenter (fiche-outil, retex, capture).

Cette démarche crée une culture réflexive et transmissible.

## ❑ 4. Témoignage augmenté : le retex conversationnel

Lors d'un atelier métier, un membre présente une interaction LLM réussie (ou non) :

- Objectif du prompt
- Résultat obtenu
- Ce qui a aidé ou bloqué
- Motif utilisé ou émergent

Ce format court (10–15 min) favorise la **démocratisation du langage de motifs**, sans expertise préalable.

## 🌐 Partage élargi : au-delà de l'équipe

- Dans une communauté interne (guilde tech, slack d'entreprise) : canal #prompt-examples
- Dans une communauté ouverte (dev.to, conférences, forums) : billet “3 motifs qui m'aident au quotidien”
- Dans une formation : intégrer les motifs comme “piliers de posture”

## Une posture à transmettre

Transmettre un motif, ce n'est pas donner une solution. C'est inviter à

l'expérimentation, à l'ajustement, à la création de sens.

Le langage de motifs est vivant. Il appelle à :

- la **curiosité** (oser tester),
- la **lucidité** (savoir ce qui ne marche pas),
- la **générosité** (partager ses trouvailles, ses ratés, ses doutes).

## ❸ En résumé

- Les motifs sont des outils d'apprentissage **par et pour la pratique**.
- Leur transmission passe par **des formats simples, incarnés, reproductibles**.
- Créer une culture de partage autour des motifs, c'est **ancrer durablement les bons usages des LLM**.

Une organisation qui apprend à partager ses motifs devient **une organisation réflexive, augmentée, résiliente**.

# Chapitre 11 — Usages de l'IA dans l'apprentissage

## L'IA comme tuteur et partenaire pédagogique

L'un des usages les plus immédiats des LLM est **l'auto-formation guidée**. Le développeur ou la développeuse peut interroger le modèle comme un mentor disponible à tout moment : pour demander une explication, une analogie, un exemple de code, ou une reformulation.

**Exemple :** – « *Peux-tu m'expliquer les closures en JavaScript comme si j'avais 12 ans ?* » – « *Montre-moi trois variantes de cette fonction, du plus simple au plus optimisé.* »

Cela permet à chacun·e d'apprendre à son rythme, de combler des lacunes rapidement, et de mieux consolider ses connaissances. Le LLM devient alors un **compagnon d'apprentissage** permanent, personnalisable et sans jugement.

De nombreuses équipes encouragent déjà cet usage comme un réflexe naturel : ne pas rester bloqué·e, mais “demander à l'IA” avant de déranger un collègue — ou au contraire, pour préparer un échange plus ciblé.

## 2. Génération assistée de documentation

La production de documentation est souvent négligée ou reportée. Avec l'IA, il devient possible de la générer **de manière incrémentale et contextuelle**, à partir :

- de la lecture d'un fichier source ;
- d'un historique de commits ou de tickets ;
- d'un échange de chat technique ;
- d'une démonstration enregistrée.

**Exemples :** – Générer automatiquement des docstrings à partir du code. – Proposer un résumé technique d'un module ou d'un ticket. – Synthétiser un document Markdown à partir d'un échange Slack ou Notion.

Ce type de documentation “sur demande” réduit la friction cognitive, et permet une mise à jour plus régulière. Il devient aussi plus facile de partager cette documentation avec des profils non techniques (PO, UX, métiers...).

## 3. Prompts comme artefacts à versionner

Un des concepts les plus innovants dans ce nouveau paradigme est celui de **prompt comme artefact documentaire**. Autrement dit : un prompt bien formulé peut devenir une *ressource à part entière*, au même titre qu'un test unitaire ou qu'un ticket Jira.

**Exemple :** Un prompt utilisé pour générer un plan de test automatisé ou un gabarit de composant peut être stocké, versionné, relu, partagé, adapté à d'autres projets.

Cela implique :

- de conserver une trace des prompts importants (dans Git, dans un wiki, dans une base de prompts) ;
- d'y associer leur contexte (besoin, objectif, contraintes) ;
- de relire ces prompts collectivement (comme une revue de code).

Certains outils émergent déjà autour de cette idée : *prompt repositories*, *prompt templates*, *prompt linters*, etc. Cela crée une culture de **transparence et de partage de la pensée design**, là où beaucoup de décisions étaient auparavant implicites.

## 4. Ateliers d'équipe, learning loops et coaching augmenté

L'IA peut aussi enrichir les dynamiques d'équipe, en nourrissant des **rituels collectifs d'apprentissage**. Voici quelques formats efficaces :

### Atelier “Prompt Clinic”

Chaque membre apporte un prompt qu'il ou elle a utilisé, et l'équipe discute de :

- sa clarté ;
- sa robustesse ;
- les résultats obtenus ;
- les possibilités d'amélioration.

Cela permet de partager des pratiques de formulation et de cultiver une posture réflexive.

### Learning Loop augmentée

Une mini-boucle d'apprentissage guidée par IA, par exemple :

1. Formulation d'un besoin flou.
2. Première réponse IA.
3. Reformulation humaine.
4. Affinement IA.
5. Documentation du processus.

L'équipe en tire un enseignement formel (nouveau motif, décision d'architecture, exemple à conserver).

### Coaching augmenté

Les coachs techniques ou agiles peuvent s'appuyer sur l'IA pour :

- reformuler des points techniques pendant les revues ;

- proposer des ressources adaptées aux profils juniors ;
- modéliser différentes stratégies de résolution d'un même problème.

Cela favorise une montée en compétences rapide, sans alourdir la charge humaine de transmission.

### **Prompt Book : un nouveau type de livrable**

De plus en plus d'équipes documentent leurs pratiques LLM dans un “*prompt book*” ou “*carnet de design dialogué*” : une collection structurée de prompts testés, commentés, adaptés à leur contexte métier. Ce format devient un **patrimoine collectif**, précieux pour l'onboarding, la mémoire projet, et la montée en compétence.

# 📁 Chapitre 12 — Documenter, archiver, capitaliser : vers une mémoire augmentée

Chaque échange avec un LLM laisse une trace. Mais si cette trace n'est ni conservée, ni structurée, ni transmise, elle s'efface. Concevoir avec un LLM, c'est aussi **prendre soin d'une mémoire nouvelle** : conversationnelle, vivante, partagée.

## Pourquoi ce chapitre ?

Les motifs que nous avons explorés naissent de situations concrètes. Mais pour les faire vivre dans le temps, ils doivent être **documentés, archivés, capitalisés**.

Ce chapitre vous propose des façons de construire une **mémoire augmentée**, individuelle ou collective, au service de la qualité, de l'apprentissage et de la transmission.

## 📁 Trois niveaux de mémoire augmentée

### 1. Mémoire d'interaction

Conserve les traces d'un échange précis avec un LLM. Utilité : rejouer, relire, apprendre de l'expérience.

Élément	Contenu typique
Prompt original	Avec contexte et intention
Réponse du LLM	Version retenue ou itération intermédiaire
Modifications humaines	Ce qui a été gardé, rejeté, modifié
Tags ou motif associé	“exploration guidée”, “miroir technique”

👉 Format proposé : Fiche `.prompt.md` ou entrée Obsidian/Notion 📁 Exemple de nommage : `2025-05-05_motif-miroir_auth-service.md`

### 2. 📁 Mémoire projet

Intègre les productions IA dans les artefacts projet. Utilité : compréhension future, relecture, audits.

Type d'objet	Exemple de documentation associée
Code généré	Commentaire avec prompt source + version du LLM
Spécification	Archive de la conversation ayant mené à une user story
Architecture	Synthèse IA comparant 2 options d'implémentation

Tests	Origine du jeu de test (généré, adapté, validé par l'équipe)
-------	--

👉 **Format proposé** : Dossier /doc/ai\_interactions/, avec prompt + réponse + retex  
 ↗ Bonus : créer une **PR augmentée**, qui explique comment l'IA a contribué

### 3. 📜 Mémoire collective

Formalise les motifs, bonnes pratiques, prompt canvas et tests d'intention utiles à l'équipe ou à la communauté.

Élément	Usage
Bibliothèque de motifs vécus	Formation, review, onboarding
Promptothèque commentée	Réutilisation, adaptation
Journal d'équipe génératif	Historique d'usage, discussion, évolution
Grammaire maison	Guide de formulation interne

👉 **Outils associés** : Miro / Notion / Gitbook / Docusaurus... ↗ Conseil : commencez petit.  
 Une page "Motifs de la semaine" suffit à démarrer.

### Exemple de mémoire vivante : un dossier “/prompts/”

```
/prompts/
2025-06-01_refactor_service.md
2025-06-03_auth_vs_oauth_comparison.md
2025-06-05_ui_a11y_review.md
```

Chaque fichier contient :

- Contexte (qui, quand, pourquoi)
- Prompt original
- Réponse choisie
- Modifications humaines
- Motifs associés
- Leçon(s) tirée(s)

Ce dossier peut être synchronisé avec Git, intégré dans les revues ou présenté lors des rétrospectives.

### Vers une architecture de la mémoire conversationnelle

Une “mémoire augmentée” n'est pas un répertoire figé. C'est :

- **Un espace de dialogue avec les futurs contributeurs**
- **Un support d'apprentissage et d'amélioration continue**
- **Un levier de confiance et de transparence**

Elle peut être **personnelle, d'équipe, ou collective**, mais elle doit toujours être :

- accessible,
- compréhensible,
- contextuelle,
- mise à jour.

## ☞ En résumé

- Documenter les échanges avec les LLM, ce n'est pas du formalisme. C'est de l'**architecture cognitive**.
- Trois niveaux à envisager : **interaction, projet, collectif**.
- Une mémoire bien organisée permet **de capitaliser sans rigidifier**.
- C'est un pilier fondamental pour transmettre, maintenir, sécuriser et apprendre.

Une mémoire augmentée, ce n'est pas une archive. C'est une **trace vivante d'un dialogue de conception**.

# Chapitre 13 — Scénarios prospectifs : vers une ingénierie conversationnelle générative

Et si demain, le développement logiciel n'était plus un processus centré sur le code, mais une série de dialogues, de validations progressives, de co-conceptions fluides entre humains et agents conversationnels ? Si certaines équipes n'étaient plus composées que de rôles de supervision, de validation et d'orchestration ? Ce chapitre explore plusieurs scénarios à la frontière du plausible, pour interroger les devenirs possibles du développement augmenté.

## 1. L'équipe “full-LLM” : un opérateur humain pour 5 agents spécialisés

Dans cette configuration, un humain ne code plus lui-même, mais orchestre un ensemble d'agents conversationnels spécialisés :

- **Un agent d'architecture** pour modéliser les choix de structure, les dépendances et les patterns.
- **Un agent de développement front-end** orienté design system, accessibilité et cohérence UX.
- **Un agent back-end** orienté APIs, performance, sécurité.
- **Un agent testeur** générant des cas de tests, vérifiant les conditions limites, construisant les mocks.
- **Un agent sémanticien** chargé de la documentation, des schémas explicatifs, et de la cohérence conceptuelle.

L'humain interagit en langage naturel avec ces agents. Son rôle ? Poser le cadre, arbitrer, valider les itérations.

### Extrait de dialogue fictif :

**Humain** : Je veux une API REST pour gérer un agenda partagé entre utilisateurs.

**Agent archi** : Je propose une architecture hexagonale, avec stockage PostgreSQL et middleware auth. D'accord ?

**Humain** : Ajoute une logique de permissions fines. Front, tu suis ?

**Agent front** : Oui, je vais générer une interface React avec Tailwind, accès conditionnel selon rôle.

**Humain** : Testeur, fais un plan de tests critiques sur les accès concurrents.

**Agent test** : Génération en cours...

**Humain** : Sémanticien, documente le schéma de droits en langage clair.

**Agent sémanticien** : Voilà une première version.

Ce type de scénario reste prospectif, mais les briques technologiques sont en voie de maturation.

## 2. Architecture générative pilotée par dialogue

Dans les approches classiques, l'architecture logicielle est figée en amont ou modifiée à coût élevé. Dans une approche augmentée, elle devient **évolutive, exploratoire, dialogique**.

Le concepteur discute avec un modèle qui :

- propose plusieurs patterns possibles selon les besoins exprimés (scalabilité, auditabilité, latence...),
- anticipe les points de friction ou d'entropie (ex. : microservices trop granulaires),
- simule des scénarios d'évolution (ajout de fonctionnalités, migration cloud, adaptation RGPD...).

**Exemple :** “Et si on devait gérer demain des utilisateurs sur mobile offline, notre architecture tiendrait-elle ?” — Le modèle peut simuler les adaptations nécessaires (caching local, synchro différée, message queues...).

Le rôle de l'architecte devient alors **scénariste de trajectoires techniques** plutôt que bâtisseur d'un plan fixe.

## 3. Le design conversationnel comme forme de développement

De plus en plus, le développement logiciel devient un dialogue : avec les utilisateurs, avec les autres membres de l'équipe, avec les IA. Le **design conversationnel** devient une compétence centrale :

- **Comment poser les bonnes questions pour faire émerger une solution ?**
- **Comment guider l'IA vers une intention précise sans être sur-spécifique ?**
- **Comment représenter visuellement un raisonnement émergent ?**

Ce design n'est plus uniquement une UX de chatbot. Il devient le **noyau de l'interaction avec les systèmes**.

### Atelier “Prompt-Design as Code”

Dans certaines équipes, les prompts structurants sont versionnés, testés et revus comme du code. On y applique des patterns de lisibilité, de modularité et de robustesse. Le prompt devient un artefact central du projet, pas un simple outil temporaire.

## Vers une nouvelle ingénierie : augmentée, exploratoire, réflexive

Ces scénarios esquisSENT les contours d'une ingénierie profondément transformée :

- **Les compétences se déplacent** de la maîtrise syntaxique vers l'anticipation, l'orchestration et la relecture critique.
- **Les rôles se fluidifient**, mêlant conception, modélisation, facilitation et test dans des cycles courts de dialogues.
- **Les outils deviennent des partenaires de raisonnement**, capables de simuler, reformuler, proposer.

Ce n'est pas tant l'IA qui remplace le développeur, que l'ingénierie qui devient **un espace de conversations raisonnées, guidées par l'intention, la rigueur et l'éthique**.

# Chapitre — Dois-je avoir honte d'utiliser l'IA dans mon métier de développeur informatique ?

Ce n'est pas la machine qui est honteuse. C'est le regard que l'on porte sur ce que l'on en fait.

## ❸ Une question qui revient souvent

« Tu fais faire ton boulot par une IA maintenant ? » « Tu codes ou tu copies/colles ce que ChatGPT te dit ? » « T'as pas l'impression de tricher ? »

Si vous avez déjà utilisé un LLM dans votre activité de développeur, il est probable que vous ayez entendu — ou pensé — ce genre de remarques. Derrière elles, une question plus large se pose : **dans quelle mesure l'usage de l'IA remet-il en cause notre légitimité professionnelle ?**

## Travailler avec une IA, ce n'est pas déléguer son métier

Utiliser un LLM ne signifie pas « sous-traiter son cerveau ». Cela signifie **articuler son expertise avec un outil génératif**. Et cela demande des compétences spécifiques :

- Cadrer l'intention et la reformuler,
- Juger de la qualité d'une réponse,
- Itérer, tester, valider,
- Adapter au contexte, au style, à l'architecture du projet,
- Documenter et transmettre ce qui a été co-produit.

Si vous utilisez l'IA sans comprendre ce qu'elle produit, vous prenez un risque technique. Si vous la comprenez, vous augmentez votre capacité à explorer, créer, décider.

## ✿✿ Ce qui pose problème, ce n'est pas l'outil — c'est l'usage

Vous ne devriez pas avoir honte d'utiliser une IA. Mais vous devriez peut-être avoir honte de :

- Intégrer une réponse sans test ni validation,
- Copier une solution que vous ne comprenez pas,
- Dissimuler l'origine d'un bloc de code généré,
- Présenter comme vôtre une idée que vous n'avez ni reformulée, ni pensée.

L'éthique de l'usage précède la technologie.

## Réhabiliter le droit à l'outillage

A-t-on eu honte d'utiliser un IDE ? A-t-on eu honte d'utiliser des bibliothèques ? A-t-on eu honte de Stack Overflow ?

Le LLM est un nouvel outil dans cette longue chaîne d'augmentation du développeur. La honte n'a pas sa place ici — la lucidité, si.

## Affirmer une nouvelle fierté professionnelle

La fierté n'est pas dans le « tout fait main ». Elle est dans la capacité à :

- formuler clairement,
- penser en interaction,
- produire du code responsable,
- documenter les choix,
- transmettre les motifs de conception,
- apprendre avec, et non malgré, les IA.

Ce livre ne vous propose pas d'être un développeur qui "ne triche pas". Il vous propose d'être un développeur qui **pense mieux avec les bons outils**.

Alors non, vous ne devriez pas avoir honte. Vous devriez être prêt à en parler, à l'assumer, à en faire un levier — de qualité, de transmission, de réflexivité.

# Conclusion : vers un manifeste du développement augmenté

Et maintenant ? Ce dernier chapitre trace les contours d'un manifeste pour un développement logiciel augmentant l'humain, et non le remplaçant.

Tout au long de cet ouvrage, nous avons exploré les mutations profondes que l'arrivée des LLM induit dans les pratiques du développement logiciel. Nous avons vu qu'il ne s'agit pas simplement d'accélérer la production de code ou d'ajouter un nouvel assistant virtuel aux outils existants, mais bien de transformer notre rapport à la conception, au langage, à l'équipe, et à l'intention.

Le **développement augmenté** n'est pas une méthode, ni une boîte magique. C'est un artisanat réinventé, fondé sur des dialogues éclairés, des motifs partagés, et une posture humble mais exigeante face à l'intelligence artificielle.

## Ce que nous retenons

- **Le LLM est un miroir des intentions.** Il amplifie la clarté, ou révèle le flou. Dialoguer avec lui, c'est dialoguer avec soi-même — ou avec l'équipe.
- **Le prompt devient une unité de conception.** Il est à la fois outil de pilotage, point d'entrée, artefact à documenter et à partager.
- **Les motifs d'interaction ont une valeur pédagogique et opérationnelle.** Ils aident à structurer les usages, à transmettre les pratiques, à éviter les pièges courants.
- **Les outils n'ont de puissance que dans un cadre collectif explicite.** Sans grille d'analyse partagée, sans rituel d'échange ou de validation, les risques d'erreurs ou d'illusions augmentent.
- **L'IA peut libérer du temps pour la qualité, l'architecture, la compréhension.** À condition que l'intention soit claire, la supervision active, et le cadre sain.

## Une posture nouvelle

Cette transition invite à **changer de rôle** : de producteur de code à concepteur de systèmes ; de technicien exécutant à partenaire d'un dialogue avec la machine ; d'architecte solitaire à membre d'un collectif augmenté.

Elle suppose d'**apprendre à apprendre autrement**, en itérant, en reformulant, en confrontant, en expérimentant. Elle valorise des compétences jusqu'ici périphériques : clarté d'expression, sens du contexte, capacité à expliciter des arbitrages ou à interroger une intuition.

## Et maintenant ?

Voici quelques propositions pour prolonger la dynamique :

1. **Documenter vos propres motifs.** Quels types de dialogue LLM utilisez-vous ? Dans quel contexte ? Avec quel succès ?
2. **Partager vos canevas et prompts.** Entre équipes, au sein d'un écosystème, dans des communs open source.
3. **Organiser des revues de prompts.** Comme on fait des revues de code, pour apprendre ensemble et améliorer la qualité de la formulation.
4. **Expérimenter de nouveaux rituels.** Un "Design Studio" avec IA, un kata de prompting, une grille de supervision éthique...
5. **Contribuer à ce langage vivant.** Ce livre n'est qu'un point de départ. Les motifs peuvent évoluer, se combiner, se décliner selon les contextes.

## Un manifeste du développement augmenté (version 0.1)

- Nous concevons avec l'IA, non à sa place.
- Nous formulons nos intentions avec précision, curiosité et exigence.
- Nous évaluons les réponses avec rigueur, esprit critique et coopération.
- Nous construisons une culture d'équipe autour de pratiques explicites et partageables.
- Nous expérimentons pour apprendre, nous partageons pour progresser, nous adaptons pour durer.

**Ce manifeste est ouvert.** Il attend vos extensions, vos reformulations, vos cas concrets. Le développement augmenté est une aventure collective : à nous de la dessiner ensemble.

## Annexes — Fiches d'outils

Après l'exploration des idées et des motifs, place à la mise en œuvre. Voici les outils que vous pouvez mobiliser, adapter et faire vivre dans vos projets dès aujourd'hui.

### Fiche-Outil 1 — Motif d'interaction LLM

À remplir pour documenter un motif observé, expérimenté ou transmis.

Élément	Description
<b>Nom du motif</b>	Intitulé court, évocateur
<b>Contexte d'usage</b>	Quand et pourquoi ce motif s'active ?
<b>Problème adressé</b>	Quelle difficulté récurrente ce motif aide-t-il à résoudre ?
<b>Forme du prompt type</b>	Exemple générique de formulation
<b>Démarche recommandée</b>	Étapes ou enchaînements d'interaction
<b>Conséquences positives</b>	Ce que permet ou renforce ce motif
<b>Variantes ou adaptations</b>	Exemples d'ajustement selon le contexte (individuel, équipe, langage...)
<b>Points de vigilance</b>	Risques, limites, pièges courants
<b>Exemples vécus</b>	Projet, situation, anecdote concrète
<b>Tags</b>	Thèmes associés (exploration, refactor, tests, doc, sécurité...)

### Fiche-Outil 2 — Design de Prompt

À utiliser pour concevoir ou améliorer un prompt dans un cas d'usage réel.

Élément	Contenu
<b>Titre ou objectif du prompt</b>	Ce que vous attendez de l'interaction
<b>Contexte donné au LLM</b>	Langage, framework, contraintes métier, niveau de l'utilisateur
<b>Tâche demandée</b>	Action ou production attendue
<b>Intentions spécifiques</b>	Pourquoi ce prompt est important, quel but il sert
<b>Format souhaité de réponse</b>	Code, texte, tableau, résumé, diagramme...
<b>Exemple de prompt</b>	

<b>Tests d'attente (facultatif)</b>	Exemples de sortie valide ou critères de succès (→ TDP)
<b>Variantes à tester</b>	Reformulations possibles, précisions à ajouter
<b>Retours d'expérience</b>	Ce qui a bien/mal fonctionné

### Fiche-Outil 3 — Test-Driven Prompting (TDP)

Pour formaliser les attentes avant de rédiger un prompt

Élément	Contenu
<b>Nom ou but du prompt</b>	
<b>Entrées fournies au LLM</b>	(ex. code source, contexte métier, exemple de données)
<b>Sortie attendue type 1</b>	Exemple de réponse idéale
<b>Sortie attendue type 2 (variante)</b>	
<b>Contraintes formelles</b>	Longueur, ton, format, style, contenu interdit
<b>Critères d'échec</b>	Ce qui rend une réponse inutilisable
<b>Prompt associé</b>	À construire en regard des tests ci-dessus

### ❖ Fiche-Outil 4 — Retex d'interaction LLM

Pour documenter une interaction significative avec un LLM (succès ou échec)

Élément	Détail
<b>Date &amp; contexte</b>	Projet, étape, objectif
<b>Prompt utilisé</b>	
<b>Réponse obtenue</b>	Résumé ou extrait
<b>Satisfaction ? (●○○○)</b>	Évaluer la pertinence ou utilité
<b>Ce qui a bien fonctionné</b>	
<b>Ce qui a été problématique</b>	(hallucination, confusion, biais, etc.)
<b>Enseignements tirés</b>	
<b>Motifs associés (si connus)</b>	

## Annexe 2 — PO augmenté : pratiquer son rôle avec l'appui d'un LLM

⌚ *Le LLM est un partenaire d'exploration, pas un pilote produit.* Cette fiche s'adresse aux Product Owners qui souhaitent intégrer l'usage des LLM dans leur quotidien, sans déléguer leur responsabilité ni brouiller leur posture.

### ✓ Objectifs

- Gagner en clarté, rapidité et structuration dans la conception fonctionnelle.
- Préparer des supports de communication (user stories, pitches, release notes...).
- Explorer plusieurs options de formulation avant de trancher.
- Interroger ou challenger un besoin métier.
- Garder la trace d'un raisonnement produit assisté.

### ⚠ Points de vigilance

- **Ne pas déléguer l'intention produit** : le LLM ne connaît ni vos utilisateurs, ni votre stratégie.
- **Relire systématiquement les propositions** : même bien tournées, elles peuvent être inexactes.
- **Ne pas injecter de données sensibles** (priorités internes, roadmap confidentielle, noms clients...).
- **Partager les prompts utiles** à l'équipe pour créer un référentiel commun.

### Postures recommandées

Situation	Motif recommandé	Posture du PO
Besoin flou ou mal exprimé	⌚ Question socratique	Clarificateur
Plusieurs options à trancher	⌚ Modèle miroir	Décideur critique
Story à rédiger ou reformuler	Spécification inversée	Rédacteur-explorateur
Définir un critère d'acceptation	Prompt piloté par les tests	Garant de la qualité
Formaliser une décision	📁 Journal de prompt	Responsable de la traçabilité

### ❖ Exemples de prompts utiles

**Exploration de formulation** « Voici une fonctionnalité que je souhaite décrire. Peux-tu me proposer trois façons différentes d'en écrire l'user story, avec des approches centrées utilisateur ? »

**Définition de critère d'acceptation** « Voici une story et son objectif. Propose-moi 3 critères d'acceptation mesurables, inspirés du format Gherkin. »

**Validation métier** « Voici une fonctionnalité envisagée. Quelles sont les questions à poser pour s'assurer de sa valeur métier réelle ? »

### 📎 Pour aller plus loin

- Partager vos prompts utiles dans un canal d'équipe.
- Organiser une revue collective des prompts de specs.
- Créer un répertoire “prompts validés” par domaine fonctionnel.

# **Et si coder ne signifiait plus écrire du code, mais converser pour concevoir ?**

L'avènement des modèles de langage (LLM) transforme radicalement la manière de penser, d'écrire et de faire vivre le logiciel. Développeur, architecte, enseignant ou coach : chacun voit son rôle évoluer vers une nouvelle forme d'artisanat, mêlant langage naturel, intelligence artificielle et design itératif.

**Ce livre propose un langage de motifs** — inspiré des travaux de Christopher Alexander — pour structurer ces nouvelles pratiques. Chaque motif décrit une situation courante, un problème typique, une solution éprouvée et ses conséquences. On y découvre comment formuler un prompt efficace, dialoguer avec un LLM pour concevoir une architecture, affiner une intention métier, évaluer la qualité d'un code généré ou encore co-écrire des tests avec précision.

**Concret, structuré, et profondément humain, cet ouvrage est à la fois :**

- un **guide pratique** pour les professionnels du développement logiciel,
- un **manifeste** pour une collaboration responsable entre humains et IA,
- un **outil d'apprentissage** pour celles et ceux qui souhaitent anticiper les mutations en cours.

Issu d'un dialogue entre un praticien chevronné et une IA de génération de texte, ce livre est aussi un exemple vivant de ce qu'il explore : une écriture augmentée, collective, en mouvement.

**Une invitation à repenser la conception logicielle comme une conversation enrichie.**