

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) et le soutien de Gemini et Copilot

Idée originale d'Olivier Azeau

Bêta-lecteurs Gowen Pottiez, Guillaume Saint-Etienne, Cindy Schlaufmann, Philippe Charrière

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

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.

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.

Chapitre 15 — Repenser les design patterns à l'ère des LLM

Quand la conception devient conversationnelle. Ce chapitre revisite les design patterns classiques à la lumière des interactions avec les LLM. Et si les motifs du code devenaient des motifs de dialogue ? Une invitation à repenser la conception logicielle non plus comme une architecture figée, mais comme un langage vivant centré sur les intentions.

Chapitre 16 — Nouveaux design patterns émergents à l'ère des LLM et des agents IA

De la génération au pilotage conversationnel. Découvrez les motifs émergents nés de la collaboration entre humains, LLM et agents spécialisés : Copilote explorateur, Spécifieur génératif, Réviseur sémantique... Une nouvelle grammaire de design où l'orchestration, la réflexivité et l'hybridation deviennent des compétences clés.

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.

Annexe 1 — 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 — TDP : Test-Driven Prompting

Écrire des prompts comme on écrit du code testable. Découvrez une méthode rigoureuse pour stabiliser et fiabiliser vos interactions avec les LLM, en définissant les attentes avant même de rédiger le prompt. Une approche inspirée du TDD, pour passer d'un prompt « qui marche une fois » à un artefact robuste, maintenable et partageable.

Annexe 3 — 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.

Annexe 4 — Développeur augmenté : étendre ses gestes avec l'appui d'un LLM

Du code à la conversation structurée. Cette annexe explore les gestes, postures et routines qui transforment le quotidien du développeur au contact des LLM : concevoir, explorer, reformuler, documenter... avec plus de clarté et de réflexivité. Un guide pour devenir éditeur de sens autant que producteur de code.

Annexe 5 — Coach agile augmenté :: enrichir ses accompagnements avec un LLM

L'IA comme partenaire de facilitation. Comment un coach agile peut-il intégrer un LLM dans ses rituels, ses observations et ses accompagnements ? Cette annexe propose des motifs adaptés, des exemples de prompts et des usages concrets pour renforcer l'intelligence collective sans jamais la court-circuiter.

Annexe 6 — Manager 3.0 augmenté : soutenir les dynamiques collectives avec un LLM

Pilotage, clarté, alignement : l'IA comme boussole conversationnelle. Cette annexe s'adresse aux managers qui veulent explorer les apports d'un LLM dans leur posture : prise de recul, alignement stratégique, soutien à la décision, ou facilitation d'équipe. Des usages concrets pour renforcer la lucidité, sans abdiquer la 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'autocomplétion.

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
Contexte	« Je travaille sur une API FastAPI en Python déployée sur AWS Lambda... »
Tâche claire	« Je veux une fonction qui valide un token JWT dans les headers HTTP. »
Contraintes	« Sans ORM, logs clairs en cas d'échec, Python 3.10. »
Intention	« Le but est que ce soit compréhensible pour un développeur junior. »
Format attendu	« 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.

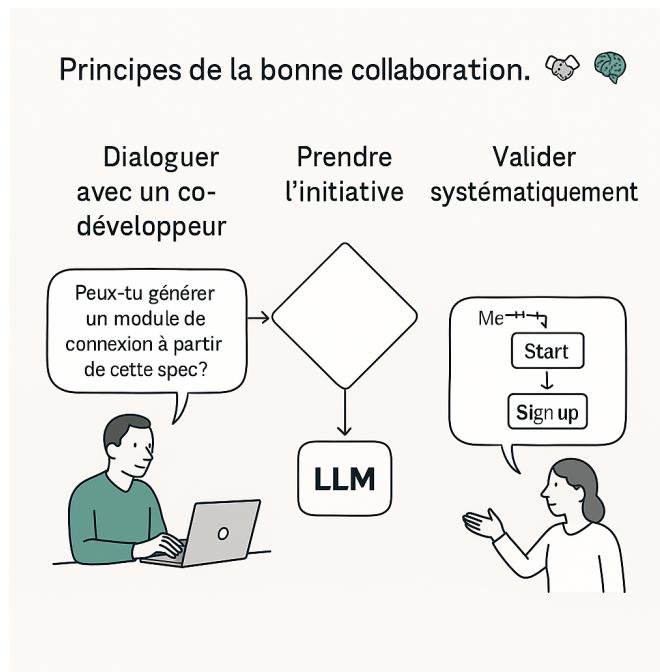
En injectant des contraintes spécifiques (taille de l'équipe, fréquence des déploiements, besoins d'évolutivité horizontale), la réponse s'affine. Le LLM devient alors un **simulateur d'options**, et le dialogue une façon d'explorer des possibles.

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

Cette bibliothèque de motifs est un outil utile pour **construire un langage de conception** avec un LLM.

➊ Motif 1 — Question Socratique : *Reformuler pour comprendre*



🎯 **Contexte** Un besoin est exprimé de façon floue, incomplète, ou imprécise — que ce soit par vous-même, un collègue, un client ou un utilisateur. Vous entrez dans une zone d'incertitude : la formulation initiale du prompt est insuffisante pour guider une réponse utile. Cela peut se produire au début d'un projet, dans une phase d'exploration, ou lors d'un échange interdisciplinaire.

🚧 **Problème** Un prompt flou génère une réponse générique, stéréotypée, ou hors-sujet. Le LLM comble les vides par des hypothèses implicites — souvent différentes de vos intentions réelles. Cela entraîne perte de temps, mauvaise orientation de la discussion ou illusion de progrès.

✓ **Solution** Adopter une posture de **questionnement socratique** : poser des **questions ciblées, progressives et ouvertes** pour affiner la compréhension de l'intention réelle. Vous invitez le modèle à vous aider à **mieux formuler votre propre besoin**. Ce faisant, vous explorez les contours de la demande avant d'attendre une réponse structurée.

Exemples de relances utiles :

- « Quels types d'erreurs souhaitez-vous capturer ? »
- « À qui est destinée l'alerte ? »
- « Quelles sont les conséquences attendues de cette action ? »
- « À quel moment dans le processus intervient ce script ? »

❖ Conséquences

- Clarifie l'intention initiale, même pour le demandeur humain.
- Enrichit le prompt au fil du dialogue.
- Déclenche un **raisonnement partagé** avec le LLM.
- Diminue le risque de mauvaise direction ou de sur-généralisation.
- Rend l'utilisateur plus conscient de ses propres besoins implicites.

💡 **Exemple d'usage** Un développeur envoie au LLM :

« *Crée un script d'alerte.* »

Réponse : trop générique, pas exploitable.

Il relance :

« *Ce script doit détecter des erreurs de facturation dans des fichiers CSV. Quels types d'erreurs puis-je surveiller ? Peux-tu me proposer des catégories ?* »

Le LLM identifie :

- Montants incohérents
- Dates invalides
- Références manquantes
- Doublons

À partir de là, le développeur reformule une demande beaucoup plus précise :

« *Génère une fonction Python qui scanne un dossier de fichiers CSV, détecte les erreurs listées ci-dessus, et envoie un rapport par mail en fin de traitement.* »

Cette démarche transforme un prompt vague en **spécification dialoguée**.

🌀 Variantes utiles

- **Pour cadrer un besoin métier :**

« *Peux-tu me poser 5 questions pour clarifier ce que je veux faire ?* »

- **Pour aider un PO ou un stakeholder :**

« *Imagine que je ne suis pas sûr de ce que je veux. Aide-moi à explorer les options à partir de mes contraintes.* »

Outils associés

- Mode *roleplay* : demander au LLM d'agir comme un UX designer, un coach agile ou un product manager.
- Canevas de clarification des besoins (cf. chapitre 9).

 **Posture recommandée** Ne cherchez pas une réponse immédiate. Cherchez la **bonne question suivante**. Ce motif vous transforme, vous aussi, en facilitateur de votre propre clarté.

Prompt-type à mémoriser

« *Aide-moi à clarifier ma demande en me posant des questions. Ne propose pas encore de solution.* »

❶ Motif 2 — Exploration guidée : *Découper pour mieux avancer*



🎯 **Contexte** Vous abordez un sujet complexe, nouveau ou flou — une architecture, un algorithme, une fonctionnalité transversale, un domaine métier inconnu. La tâche paraît vaste ou informe. Vous sentez que vous avez besoin d'un **plan d'attaque** pour avancer étape par étape.

🚧 **Problème** Le prompt initial mène à une réponse trop large, confuse ou superficielle. Vous recevez une explication générique, sans hiérarchisation des priorités ni découpage utile. Le modèle cherche à répondre à tout... sans résoudre rien de manière exploitable. Résultat : surcharge cognitive, dispersion, perte de temps.

✅ **Solution** Utiliser le LLM comme **facilitateur de structuration**. Lui demander explicitement de proposer un **découpage progressif du sujet** en étapes, catégories, niveaux d'analyse ou zones fonctionnelles. Vous ne demandez pas encore de solution, mais une **carte du territoire**.

Exemples de prompts :

- « Quelles grandes étapes pour concevoir ce module ? »
- « Peux-tu proposer un plan d'implémentation en plusieurs phases ? »
- « Découpe cette problématique en sous-problèmes techniques. »
- « Quels aspects métier devrais-je explorer en priorité ? »

➡️ Conséquences

- Réduction de la complexité perçue.
- Meilleure priorisation des tâches.
- Approche plus itérative et incrémentale.
- Découverte d'aspects non envisagés au départ.
- Meilleur alignement entre technique et fonctionnel.

 **Exemple d'usage** Deux développeurs doivent créer un module de traitement de factures dans un ERP. Prompt initial :

« Comment concevoir ce module ? »

Réponse : longue, dense, difficile à exploiter. Ils reformulent :

« Peux-tu proposer un découpage fonctionnel et technique pour construire ce module ? »

Le LLM répond :

- Identifier les sources de données (clients, fournisseurs).
- Définir les règles de validation métier.
- Structurer les statuts de traitement.
- Intégrer les notifications.
- Gérer les cas d'erreur.
- Prévoir les exports comptables.

Le découpage devient la base d'un backlog, d'un plan de MVP, et d'un dialogue structuré avec le Product Owner. Le LLM agit ici comme **médiateur d'avancement**.

Variantes utiles

- **Exploration en entonnoir** : demander un plan général → zoomer sur une étape → détailler chaque sous-étape.
- **Exploration multi-angles** : demander un découpage par rôle (technique, fonctionnel, UX), ou selon différentes priorités (coût, impact, risque).
- **Exploration critique** : demander les étapes les plus risquées, ou celles qui méritent un POC.

Outils associés

- Templates de plan d'implémentation (type RICE, MoSCoW).
- Mindmaps générées à partir de la réponse (via outil visuel).
- Utilisation du LLM comme *Project Planner* ou *architecte conversationnel*.

 **Posture recommandée** Acceptez de **ne pas chercher à tout résoudre d'un coup**.

Appuyez-vous sur le LLM pour **orchestrer une progression** : cartographier avant de coder. Vous devenez facilitateur de focus, plutôt que chercheur de solutions immédiates.

Prompt-type à mémoriser

« Je travaille sur [sujet]. Propose une décomposition en étapes concrètes et progressives, pour m'aider à structurer ma démarche. »

❸ Motif 3 — Spécification inversée : *Remonter aux intentions à partir du code*



🎯 **Contexte** Vous devez comprendre un code existant, souvent ancien, mal documenté, ou écrit par quelqu'un d'autre. Vous arrivez *après* la conception. Il n'y a pas de user stories, de documentation claire, ni d'intention explicite. Vous devez pourtant refactorer, auditer, tester, ou réexpliquer ce code.

🚧 **Problème** Le code vous montre *comment* une chose est faite, mais pas *pourquoi*. Sans les intentions d'origine, les contraintes métier ou les hypothèses implicites, vous êtes obligé de deviner. Cela rend la tâche longue, risquée, frustrante.

✓ **Solution** Utiliser le LLM comme **détecteur d'intention rétroactif**. Lui soumettre des portions de code, et lui demander de :

- reformuler les intentions fonctionnelles implicites,
- expliciter les règles métier,
- suggérer les user stories probables,
- identifier les hypothèses ou présupposés du développeur initial.

Exemples de prompts :

- « Que fait ce code ? »
- « Quelles règles métier cela semble-t-il implémenter ? »
- « Quelle user story pourrait correspondre à ce bloc de code ? »
- « Quelles hypothèses implicites sur les données ou le contexte ce code semble-t-il faire ? »

Conséquences

- Raccourcit l'analyse d'un code inconnu.
- Produit une **documentation rétroactive**.
- Fait émerger des biais ou angles morts.
- Sert de support à la revue de code, à la transmission ou à la refonte.
- Réconcilie code et métier, implémentation et intention.

 **Exemple d'usage** Lors d'un audit, une équipe hérite d'un module PHP de 800 lignes, sans test ni doc. Au lieu d'une lecture ligne à ligne, elle découpe le fichier en blocs logiques et utilise ce prompt :

« Quelles règles métier ce bloc semble-t-il implémenter ? »

Le LLM détecte :

- La détection de doublons,
- Le contrôle de TVA,
- L'arrondi conditionnel,
- Des cas particuliers non mentionnés dans la doc.

Ce travail itératif permet de reconstruire les intentions d'origine, de documenter les cas d'usage, et de planifier une refonte sans tout casser.

Variantes utiles

- **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).

- **Déduction d'hypothèses implicites**

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.

- **Contrat d'interface implicite**

Demandez au LLM :

« 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.

Outils associés

- Intégration dans IDE via plugin d'analyse augmentée.
- Prompt-routine de revue de code (cf. chapitre 9).
- Documentation générée à partir du code source, enrichie par LLM.

 **Posture recommandée** Le LLM ne remplace pas votre lecture du code, il **l'oriente**. Utilisez ses propositions comme **hypothèses de travail**, pas comme vérité. Croisez avec votre intuition, les tests existants, les retours métier.

Prompt-type à mémoriser

« Voici une fonction sans documentation. Peux-tu expliciter ce qu'elle fait, pourquoi, et quelles hypothèses elle semble faire ? »

➊ Motif 4 — Modèle miroir : *Comparer pour éclairer un choix*



🎯 **Contexte** Vous hésitez entre plusieurs solutions possibles : deux architectures, deux approches algorithmiques, deux styles de code, deux outils. L'équipe discute, mais le débat reste flou ou biaisé. Vous avez besoin de prendre du recul pour décider *en conscience*, et pas par réflexe ou préférence personnelle.

🚧 **Problème** Le LLM répond souvent avec *une seule solution* par défaut. Or, dans les situations complexes, il est plus utile de **comparer plusieurs options** que d'en générer une « réponse-type ». Sans confrontation d'alternatives, on risque de s'en tenir à une première bonne impression... sans voir les conséquences.

✅ **Solution** Utiliser le LLM comme **miroir comparatif** : lui demander explicitement de produire plusieurs variantes d'une solution, puis de les comparer selon des critères définis (lisibilité, performance, maintenabilité, UX...). Cela transforme la réponse en **analyse dialectique**, qui éclaire la décision.

Exemples de prompts :

- « Propose deux implémentations de cette fonction : l'une impérative, l'autre fonctionnelle. Compare-les. »
- « Donne trois options d'architecture et leurs avantages/inconvénients selon nos contraintes. »
- « Compare React et Svelte pour ce type de projet. »

Conséquences

- Favorise l'analyse critique au lieu du mimétisme.
- Explicite les critères de choix.
- Aide à la décision collective, surtout dans un contexte d'équipe.
- Réduit le biais de confirmation ou d'autorité.
- Sert de support à la documentation des décisions.

 **Exemple d'usage** Dans un projet de refonte de système de paiement, l'équipe hésite entre :

- Une architecture orientée événements avec Kafka
- Une architecture REST synchrone plus classique

Le prompt devient :

« *Compare ces deux options pour un système haute disponibilité avec 100 transactions/s. Quels sont les compromis ?* »

Le LLM souligne que :

- Kafka est plus résilient mais plus complexe à moniturer,
- REST est plus simple à tester mais moins robuste aux pics de charge.

La discussion s'appuie sur ces éléments pour prendre une **décision argumentée**, et pas simplement « parce qu'on a toujours fait comme ça ».

Variantes utiles

- **Miroir de styles** : comparer style impératif vs fonctionnel, orienté objet vs déclaratif.
- **Miroir de paradigmes** : polling vs event-driven, synchronisme vs asynchronisme.
- **Miroir d'outils** : frameworks front, moteurs de base de données, bibliothèques de tests, etc.
- **Miroir UX** : comparer deux messages d'erreur, deux parcours utilisateur.

Outils associés

- Grille de comparaison à co-construire avec le LLM.
- Tableau à double entrée : options × critères.
- Intégration possible dans une documentation de choix d'architecture (ADR).

 **Posture recommandée** Demandez *plusieurs options* avant de creuser une seule. Faites du LLM un **stimulateur de divergence raisonnée**. Il ne décide pas à votre place — il éclaire le chemin.

Prompt-type à mémoriser

« *Propose plusieurs alternatives pour ce besoin, puis compare-les selon ces critères : [X, Y, Z].* »

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



🎯 **Contexte** Le LLM a produit une réponse satisfaisante — un code, une solution technique, une recommandation. Tout semble correct... mais une forme de doute persiste. Est-ce vraiment robuste ? La réponse couvre-t-elle tous les cas ? Le raisonnement tient-il dans les cas extrêmes ?

🚧 **Problème** Le modèle donne souvent une solution « idéale » ou typique, qui **masque les cas limites ou les situations d'échec**. Le développeur peut être tenté de faire confiance par défaut. Pourtant, sans mise à l'épreuve, on risque de déployer une solution fragile, biaisée ou naïve.

✓ **Solution** Interroger la réponse **par la négation** : demander un **contre-exemple**, une situation où la solution échoue, devient inefficace ou produit un effet inattendu. Cela révèle les **limites implicites** du raisonnement et affine la compréhension de ce que la solution couvre — ou pas.

Exemples de prompts :

- « Dans quel cas cette solution pourrait échouer ? »
- « Peux-tu proposer un exemple de données qui poserait problème ? »
- « Et si le fichier est vide ? Si la connexion échoue ? Si l'utilisateur n'est pas authentifié ? »
- « Quelle hypothèse implicite, si elle est fausse, rend cette solution invalide ? »

📍 **Conséquences**

- Détection précoce des cas limites.
- Meilleure robustesse de la solution proposée.
- Formation d'une posture critique chez le développeur.
- Réduction des effets de bord ou des surprises en production.
- Enrichissement du prompt initial si besoin (cf. motif 6).

 **Exemple d'usage** Un étudiant demande au LLM d'implémenter l'algorithme de Dijkstra en JavaScript. La solution paraît correcte. Il relance avec :

« *Et si le graphe contient des cycles négatifs ?* »

Le LLM répond :

« *Dijkstra n'est pas adapté à ce cas. Il faudrait utiliser Bellman-Ford, qui gère les poids négatifs.* »

Cette simple relance transforme une session de génération en **moment d'apprentissage algorithme**, en rendant visible une hypothèse invisible.

Variantes utiles

- **Test de bord** : « Et si le tableau est vide ? Si une donnée est nulle ? »
- **Stress test** : « Et si 10 000 utilisateurs accèdent à ce module en même temps ? »
- **Contre-règle métier** : « Quelle situation métier invaliderait cette règle ? »
- **Débat simulé** : « Peux-tu simuler l'avis d'un développeur qui critique cette solution ? »

Outils associés

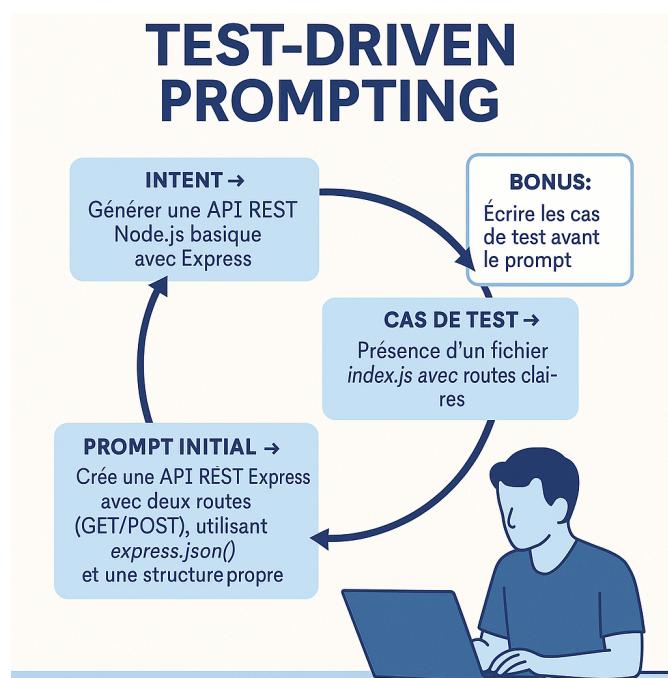
- Table de tests d'acceptation enrichie par le modèle.
- Utilisation combinée avec la génération de jeux de tests (cf. motif 3).
- Pairing augmenté : un développeur joue l'avocat du diable avec le LLM.

 **Posture recommandée** Ne te satisfais pas de la « bonne réponse » en apparence. Adopte une **posture scientifique** : falsifier, tester, pousser la logique jusqu'à ses bords. C'est ainsi que le LLM devient un **partenaire critique**, et non un automate flatteur.

Prompt-type à mémoriser

« *Donne un cas qui fait échouer cette solution. Qu'est-ce que cela révèle sur ses limites ?* »

❶ Motif 6 — Prompt piloté par les tests TDP : Définir les attentes avant d'écrire



🎯 **Contexte** Vous voulez concevoir un prompt que vous pourrez réutiliser, partager ou intégrer dans un outil. Mais les résultats du LLM sont trop variables, parfois hors sujet, parfois très bons... sans que vous compreniez pourquoi. Il devient difficile de stabiliser l'usage.

🚧 **Problème** Le prompt est écrit à l'intuition, sans cadre clair. Les attentes ne sont pas explicitées. Résultat : le modèle improvise, et ses réponses sont **inconsistantes**. Vous corrigez a posteriori, au lieu de piloter la production en amont.

✓ **Solution** Adopter une démarche inspirée du **Test-Driven Development** (TDD) : avant de rédiger le prompt, **définir les attentes** que la réponse doit satisfaire. Cela peut inclure :

- des exemples concrets de sortie attendue (mock output),
- des critères de structure, de style ou de contenu,
- des contraintes explicites de format, de ton, de longueur.

Ensuite, **écrire le prompt pour qu'il satisfasse ces critères**. Puis le tester. Puis l'affiner. Vous concevez le prompt **comme une unité fonctionnelle à valider**.

Exemples de critères préalables :

- « La réponse doit tenir en moins de 3 phrases. »
- « Utiliser un ton empathique mais professionnel. »
- « Ne jamais mentionner d'excuses juridiques. »
- « Commencer par une reformulation de la demande client. »

✖ Conséquences

- Vos prompts deviennent **précis, stables et réutilisables**.

- Vous gagnez du temps à long terme.
- Vous pouvez partager des prompts testés dans une équipe ou un outil.
- Vous transformez le prompt en **artefact d'ingénierie** à part entière.
- Vous détectez les limites de ce que le LLM peut ou ne peut pas bien faire.

 **Exemple d'usage** Une équipe crée un assistant conversationnel pour répondre à des tickets clients. Prompt initial :

« *Rédige une réponse empathique au client.* »

Résultat : trop long, trop vague, parfois juridiquement risqué. Ils décident de poser ces attentes **avant** :

- 2 à 3 phrases max,
- Pas d'excuses juridiques,
- Rassurer sans promettre,
- Adapter le niveau de langage au client identifié.

Ils ajustent ensuite le prompt jusqu'à ce qu'il **produise des réponses qui passent ces tests**. Le prompt est ensuite versionné, partagé, intégré dans un outil.

Variantes utiles

- **TDP visuel** : créer un exemple de sortie attendue, et demander au modèle de « coller à cet exemple ».
- **TDP collaboratif** : faire définir les attentes par plusieurs rôles (PO, UX, support, tech).
- **TDP embarqué** : inclure les critères de test *dans le prompt lui-même* :

« *La réponse doit tenir en 3 phrases max, rester neutre, et finir par une question ouverte.* »

Outils associés

- Librairie de prompts versionnés (Markdown ou Notion).
- Fichiers de test (exemples de prompts + sorties attendues).
- Evaluation semi-automatisée de réponses (score sur critères définis).

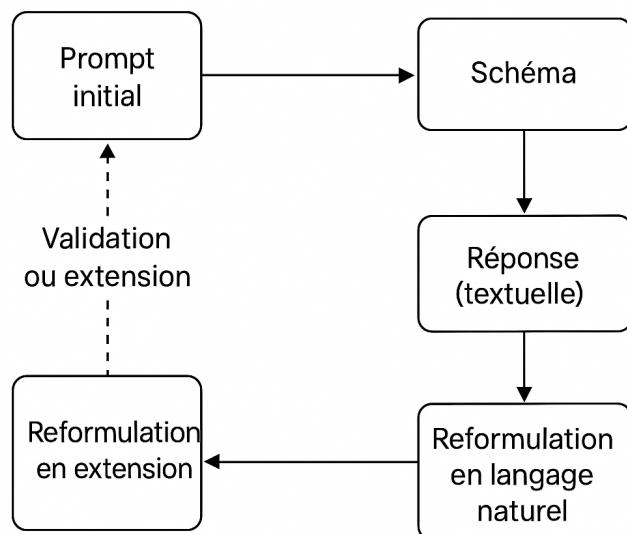
 **Posture recommandée** Concevez vos prompts **comme du code** : testables, maintenables, évolutifs. Ne vous contentez pas d'un prompt qui « marche une fois » : visez la robustesse. Cela rend vos pratiques plus pro, plus partageables, plus fiables.

Prompt-type à mémoriser

« *Voici un exemple de réponse attendue. Peux-tu formuler un prompt qui produise ce type de sortie de façon cohérente ?* »

❸ Motif 7 — Reformulation visuelle : Clarifier par la représentation

Reformulation Visuelle



🎯 **Contexte** Le LLM propose une solution textuelle : une architecture, un algorithme, un processus, une organisation de code. La réponse est intéressante, mais dense ou ambiguë. Vous soupçonnez qu'il manque des liens, que des parties sont floues ou que des incohérences se cachent dans la linéarité du texte.

🚧 **Problème** Le langage naturel masque parfois les **zones d'ombre** : raccourcis logiques, interfaces mal définies, modules manquants, étapes implicites... La solution semble complète mais elle **n'est pas vérifiée structurellement**. Sans visualisation, on risque une adhésion naïve ou une implémentation bancale.

✅ **Solution** Transformer la proposition textuelle du LLM en **schéma visuel** : diagramme de composants, flux, tableau structuré, carte mentale, etc. Puis **reformuler ce schéma en langage naturel**, et le soumettre à nouveau au modèle pour validation, critique ou enrichissement.

Exemple de boucle :

- Demande initiale → réponse textuelle du LLM
- Visualisation manuelle (draw.io, tableau, carte mentale...)
- Reformulation textuelle structurée de ce que vous avez compris
- Nouveau prompt au LLM : « Voici ma compréhension. Est-ce cohérent ? Que manque-t-il ? »

❖ Conséquences

- Fait émerger les **incohérences logiques** plus tôt.
- Facilite la validation collective dans l'équipe.
- Favorise une meilleure **appropriation humaine** de la solution.
- Stimule un échange réflexif entre humain et IA.

- Développe une compétence clé : **modéliser pour comprendre**.

 **Exemple d'usage** Le LLM propose une architecture pour un système de notifications multicanal. Un développeur la transforme en diagramme de composants :

- gestionnaire d'alerte,
- module de priorisation,
- file d'attente,
- envois webhook/email,
- Redis pour le cache.

Il reformule :

« *Si j'ai bien compris, l'alerte arrive dans un gestionnaire, qui la classe, la stocke, puis la transmet. Redis sert de cache. Est-ce juste ? Que faudrait-il ajouter ?* »

Le LLM répond :

« *Il manque un mécanisme de gestion des échecs d'envoi. Vous pourriez ajouter une file de retry avec journalisation.* »

Cette boucle de reformulation **renforce la robustesse** de la solution.

Variantes utiles

- **Tableau à double entrée** : rôles × responsabilités, modules × dépendances.
- **UML léger** : diagramme de classes, de séquence, d'activités.
- **Carte mentale** : utile pour explorer des fonctionnalités ou des flux.
- **Schéma manuscrit + transcription** : dessiner sur papier, puis demander au LLM de l'exprimer en mots.

Outils associés

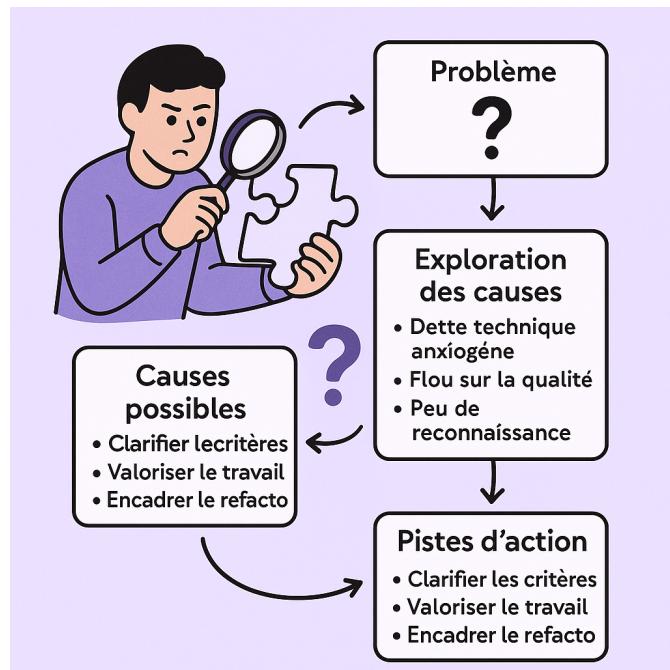
- Outils de modélisation (draw.io, Whimsical, Excalidraw...).
- Schéma intégré dans les prompts suivants : « *Voici un schéma, décris-le comme si tu l'avais proposé.* »
- Rituels d'équipe : validation d'architecture augmentée par schématisation + IA.

 **Posture recommandée** Ne restez pas dans le flou textuel. Passez par la **visualisation pour clarifier, valider, enrichir**. Le dessin ou la structure posée est une **forme de dialogue** en soi — avec soi-même, avec les autres, avec le modèle.

Prompt-type à mémoriser

« *Voici une reformulation textuelle de mon schéma. Peux-tu vérifier si elle est cohérente avec ta proposition initiale, et proposer des améliorations ?* »

● Motif 8 — Soin systémique : *Investiguer les causes profondes d'un problème*



🎯 **Contexte** Un problème persiste dans un projet ou une équipe. Il peut s'agir d'un bug récurrent, d'une démotivation latente, d'un retard accumulé, ou d'une tension interpersonnelle. Ce n'est pas simplement un défaut technique ou organisationnel isolé : **quelque chose coince en profondeur**, mais sans cause évidente.

🚧 **Problème** Le réflexe est souvent de chercher une solution rapide, locale, technique. Or, en s'arrêtant au symptôme visible, on risque de **passer à côté des vraies causes** — souvent multiples, croisées, systémiques. Le LLM, s'il est mal sollicité, proposera des rustines plutôt qu'un diagnostic structuré.

✓ **Solution** Mobiliser le LLM comme **partenaire d'investigation systémique**. Ne pas lui demander une solution directe, mais l'aider à **vous aider à creuser** :

- explorer les causes possibles d'un problème,
- croiser les angles de vue (technique, humain, organisationnel),
- proposer des pistes d'action ciblées et cohérentes avec la cause réelle.

Exemples de prompts :

- « Voici un problème récurrent dans l'équipe. Quelles causes possibles vois-tu, côté technique, relationnel, process ? »
- « Peux-tu me guider dans une session de type 5 Pourquoi / 9 Pourquoi pour en explorer la racine ? »
- « Propose un arbre logique des causes et sous-causes. »



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...).* »

❖ Conséquences

- Permet de **poser un diagnostic partagé** avant d'agir.
- Prévient les actions prématurées ou mal orientées.
- Favorise l'intelligence collective et la réflexivité.
- Crée des solutions mieux ancrées dans le réel.
- Apporte de la lucidité là où règne parfois l'agitation.

 **Exemple d'usage** Une équipe ressent une **démotivation diffuse** autour d'un module critique. Prompt initial :

« Comment remotiver l'équipe ? »

Réponses : un peu génériques (célébrer les victoires, changer d'environnement...). Le Scrum Master reformule :

« Pourquoi cette démotivation, selon toi ? Peux-tu explorer plusieurs causes possibles, en croisant les dimensions technique, humaine et organisationnelle ? »

Le LLM propose :

- dette technique anxiogène,
- flou sur les critères de qualité,
- manque de reconnaissance des efforts.

L'équipe creuse ces causes une à une, et en tire **trois actions concrètes** : clarification des règles de qualité, rituel de reconnaissance des avancées, refactoring progressif encadré.

Variantes utiles

- **Arbre des causes** : diagramme arborescent des causes/symptômes.
- **Multi-perspectives** : demander au LLM d'analyser la situation selon le point de vue d'un dev, d'un PO, d'un manager.
- **Hypothèses contradictoires** : pousser le modèle à générer plusieurs explications différentes d'un même symptôme.

Prompt-type : « Donne-moi trois hypothèses opposées sur les causes probables de cette situation. »

Outils associés

- Modèles d'analyse systémique (5 Pourquoi, 9 Pourquoi, Ishikawa, diagrammes de boucle causale).
- Canevas d'enquête partagée avec le LLM comme facilitateur de questionnement.
- Revue de sprint augmentée par exploration des causes racines (cf. chapitre 8).

 **Posture recommandée** Refuser la précipitation. Prendre le temps de **comprendre avant d'agir**. Solliciter l'IA non comme oracle, mais comme **co-investigateur systémique**. Cela demande écoute, nuance, et capacité à laisser surgir des causes non techniques dans un monde souvent technique.

Prompt-type à mémoriser

« Voici un problème qui revient souvent. Peux-tu m'aider à en explorer les causes racines selon plusieurs angles, sans proposer tout de suite une solution ? »

Pour conclure ce chapitre — Les motifs du dialogue

Résumé

Ce chapitre vous a proposé de **changer de regard sur l'interaction avec un LLM** : non plus comme une série de requêtes isolées, mais comme un **langage de conception structuré**, articulé autour de motifs conversationnels.

Vous y avez découvert :

- Ce qu'est un **motif de dialogue** : un problème récurrent, une réponse type, une conséquence attendue,
- Une première collection de motifs, comme la **Question socratique**, l'**Exploration guidée**, le **Contre-exemple** ou encore le **Prompt piloté par les tests**,
- La manière dont ces motifs s'activent en situation, dans un **dialogue pensé comme un outil de conception**.

Ces motifs ne sont ni des recettes, ni des règles : ce sont des **formes d'usage vivantes**, qui se cultivent, s'adaptent, se combinent.

Fiche récapitulative — Les 8 premiers motifs du langage LLM-Assisted Software Design

 Motif	 Finalité principale	 Geste clé	 Posture recommandée	 Prompt-type
1. Question Socratique	Clarifier un besoin flou	Poser des questions ciblées	Se mettre en position de facilitateur de sa propre pensée	« <i>Aide-moi à clarifier ma demande en me posant des questions.</i> »
2. Exploration guidée	Structurer un sujet complexe	Demander un découpage ou un plan	Explorer sans chercher à résoudre d'un coup	« <i>Propose un plan en étapes pour aborder ce sujet.</i> »
3. Spécification inversée	Comprendre du code sans doc	Inférer les intentions à partir du code	Interpréter, reformuler, confronter	« <i>Quelles règles métier ce code implémente-t-il ?</i> »
4. Modèle miroir	Comparer des options avant un choix	Générer plusieurs solutions puis les comparer	Ne pas se précipiter, ouvrir l'espace de décision	« <i>Propose 2 ou 3 options et compare-les sur ces critères...</i> »
5. Contre-exemple	Tester la solidité d'une solution	Demander un cas où ça échoue	Adopter une posture critique, tester la limite	« <i>Quelle situation rend cette solution invalide ?</i> »

6. Prompt piloté par les tests	Stabiliser la qualité des réponses	Définir les attentes avant d'écrire	Concevoir le prompt comme un artefact testable	« Voici un exemple de sortie attendue. Que faut-il inclure dans le prompt ? »
7. Reformulation visuelle	Détecter les flous dans une solution textuelle	Représenter sous forme visuelle puis reformuler	Clarifier en dessinant, dialoguer à partir du schéma	« Voici ma compréhension sous forme structurée. Que manque-t-il ? »
8. Soin systémique	Identifier les causes profondes d'un problème	Enquêter par questionnement causal	Suspendre l'action pour investiguer en profondeur	« Peux-tu m'aider à explorer les causes racines de ce problème ? »

🛠 Comment l'utiliser ?

- En **revue d'interaction avec un LLM** : repérez quel motif correspond à votre situation actuelle.
- En **atelier d'équipe** : proposez une situation → faites choisir un motif → expérimentez.
- En **coaching / mentoring** : formez aux gestes réflexifs associés à chaque motif.
- En **documentation de pratique** : ajoutez à vos templates d'atelier, vos miro ou vos outils de facilitation augmentée.

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



Concevoir avec un LLM, ce n'est peut-être pas seulement une question d'outillage. Cela pourrait représenter un changement plus profond dans la manière dont les rôles s'exercent, se redéfinissent... ou s'adaptent en continu.

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 devient pertinent de s'interroger sur ce que cette nouvelle pratique modifie dans la vie des équipes.

Quelques questions émergent naturellement :

- Est-ce que de **nouveaux rôles** apparaissent réellement ?
- Quelles **compétences prennent de l'importance**, même sans être nouvelles ?
- Les **rôles traditionnels** changent-ils de nature, ou seulement de modalités d'expression ?

Ces questions ne visent pas à produire une réponse définitive, mais à **ouvrir un espace d'observation**. Les transformations sont encore en cours, souvent silencieuses, parfois invisibles dans les grilles de poste. Ce chapitre propose d'en cartographier quelques lignes de force.

Deux grilles de lecture

Les rôles fonctionnels

Ce sont les rôles formels dans une équipe : développeur·euse, PO, tech lead, coach, etc. Ils s'inscrivent dans une organisation explicite, observable.

Les postures conversationnelles

Ce sont les manières d'interagir avec le LLM : poser des questions, filtrer, reformuler, explorer... Elles sont plus diffuses, mais tout aussi déterminantes dans un environnement augmenté.

Ces deux dimensions ne s'excluent pas. Un·e développeur·euse peut adopter tour à tour des postures de **concepteur**, de **curateur**, de **testeur d'hypothèses** — selon le moment, le contexte, l'intention.

Transformation des rôles existants

Voici une lecture possible — et évolutive — des glissements observés :

Rôle	Évolutions possibles avec l'usage des LLM
Développeur	De producteur de code à concepteur de dialogues structurés
Tech Lead	Devient garant du sens dans les chaînes de raisonnement hybrides
PO / PM	Peut gagner en autonomie sur la formulation d'hypothèses métier
UX / UI	Explore des variantes ou des idées de parcours de façon rapide
Coach agile	Introduit l'IA dans les pratiques réflexives ou les rétrospectives
Testeur / QA	Génère et évalue des jeux de tests à partir de specs ou de prompts
Formateur	Utilise le LLM comme simulateur de dialogue ou base de cas concrets

Ces évolutions ne sont pas automatiques. Elles dépendent fortement du contexte, de la culture d'équipe, de l'intérêt individuel. L'appropriation reste inégale et expérimentale.

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).

Cela suppose :

- de **maîtriser l'ambiguïté du langage naturel**,
- d'**exercer un regard critique sur les suggestions automatiques**,
- de savoir **transformer un résultat brut en artefact compréhensible, documenté, maintenable**.

Cette posture d'éditeur n'est pas nécessairement nouvelle — mais elle prend une **place plus centrale** dans un environnement augmenté.

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

Ces postures sont souvent **transversales aux rôles**. Elles peuvent coexister, se combiner, évoluer.

⌚ 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

Ces compétences peuvent s'apprendre, s'observer, se cultiver. Elles mériteraient peut-être d'être mieux valorisées.

Illustrations de terrain

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.

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 rôles sont encore marginaux. Ils posent aussi des questions : **faut-il nommer ce qui peut rester diffus ?** Peut-être s'agit-il moins d'ajouter des postes que de **reconnaître les compétences émergentes là où elles s'exercent déjà**, souvent de manière invisible.

En synthèse ouverte

- Les LLM **n'imposent pas de nouveaux rôles**, mais rendent possibles de **nouvelles postures**.
- Les équipes gagneraient à **observer comment elles s'approprient ces outils**, à leur manière.
- L'enjeu est moins de formaliser un nouvel organigramme que de **créer un espace pour explorer, nommer, transmettre** ce qui change dans les gestes du quotidien.

Le dialogue entre humains et IA n'est pas figé. Il se négocie, se cultive, s'apprend. C'est peut-être là que réside le cœur des compétences de demain.

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, et le niveau de maturité. Ce chapitre propose une **carte de ces usages**, à la fois pour mieux s'orienter et pour enrichir sa pratique.

Pourquoi ce chapitre ?

Nous avons exploré :

- des **motifs conversationnels** (Chapitre 3) — les gestes de base de l'interaction avec un LLM,
- des **rôles et postures** qui émergent dans les équipes (Chapitre 4) — les transformations en cours.

Il est maintenant temps de **connecter ces dimensions au terrain** : à ce que l'on fait concrètement avec un LLM, dans des situations précises.

L'objectif n'est pas d'exhaustivement modéliser tous les cas d'usage, mais d'**offrir une boussole** : pour reconnaître où l'on est, choisir un motif adapté, et évoluer vers des pratiques plus fluides et conscientes.

Typologie des situations

Voici une première **typologie de six situations-types** fréquemment rencontrées dans le travail logiciel augmenté par un LLM. Chaque situation est décrite par :

- une **intention centrale** (ce que l'on cherche à faire),
- des **postures activées** (comment on interagit avec le LLM),
- des **motifs associés** (les gestes conversationnels les plus utiles).

 Situation	 Intention principale	 Posture(s) activée(s)	 Motifs typiques
Exploration	Découvrir un domaine, une techno, une approche	Explorateur, apprenant	Exploration guidée, Modèle miroir
Cadrage	Clarifier un besoin flou ou implicite	Formulateur, facilitateur	Question socratique, Décomposition, Spéc. inv.
Refactorisation	Améliorer un existant	Analyste, critiqueur	Spécification inversée, Contre-exemple
Documentation	Générer ou reconstruire du sens	Synthétiseur, documentaliste	Spécification inversée, Résumé ciblé, Relecture

Validation	Vérifier une solution ou un raisonnement	Curateur, relecteur	Prompt piloté par les tests, Contre-exemple
Co-conception	Créer à plusieurs avec un LLM comme partenaire	Facilitateur, co-designer	Miroir, Clarification, Synthèse, Exploration

Ces situations ne sont ni exclusives ni rigides. Une même activité peut traverser plusieurs zones : on commence par explorer, on clarifie, on valide, on documente. C'est **un parcours, pas une case à cocher.**

Situation-type 1 — Exploration

Contexte : un développeur fullstack découvre le pattern CQRS, qu'il n'a jamais utilisé.

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

- *Exploration guidée* : pour cadrer l'apprentissage par étapes
- *Contre-exemple* : pour mettre à l'épreuve la compréhension
- *Modèle miroir* : comparaison CQRS vs CRUD pour situer les usages

Ici, le LLM devient un **tuteur patient et adaptable**, qui répond au rythme de la découverte.

Situation-type 2 — Cadrage flou

Contexte : une équipe reçoit une demande métier très vague, avec des fragments d'intention mais aucune user story claire.

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

- *Question socratique* : pour affiner ce qui manque
- *Spécification inversée* : reconstituer des règles implicites
- *Reformulation visuelle ou par test* : pour stabiliser l'intention

Dans ce type de situation, le LLM aide à **transformer du flou en structure**, à condition d'un guidage progressif.

Situation-type 3 — Refactorisation guidée

Contexte : une fonction ancienne, non testée ni commentée, doit être réécrite sans en casser la logique.

Posture : critiqueur, nettoyeur **Prompt** : « Que fait ce code ? Propose une version plus lisible avec tests associés. » **Motifs activés** :

- *Spécification inversée* : pour inférer la logique métier
- *Contre-exemple* : pour tester les limites ou bugs potentiels
- *Modèle miroir* : pour proposer plusieurs styles ou approches

L'accent est ici sur la **rétro-ingénierie assistée** : comprendre avant de modifier.

Situation-type 4 — Co-conception

Contexte : deux devs imaginent ensemble l'architecture d'un nouveau module, en dialogue avec un LLM.

Posture : facilitateur, co-concepteur

Prompts enchaînés :

- « Quels sont les patterns possibles pour ce type de traitement ? » →
- « Compare event-driven et pub/sub dans ce cas précis. » →
- « Aide-nous à rédiger un plan d'implémentation en trois étapes. »

Motifs activés :

- *Exploration guidée*
- *Modèle miroir*
- *Clarification progressive*
- *Synthèse assistée*

Le LLM agit ici comme **surface de pensée partagée**, soutenant un dialogue humain étendu.

Une carte vivante, pas une grille figée

Ce que cette typologie révèle, ce n'est pas une méthode, mais un **champ d'interactions possibles**. Elle peut devenir :

- un **outil réflexif individuel** : "Dans quelle situation suis-je ? Quel motif serait utile ?"
- un **cadre d'atelier collectif** : pour cartographier les usages de l'équipe et les enrichir
- un **levier d'apprentissage progressif** : en rendant visibles les zones encore peu explorées

Certaines équipes tiennent à jour leur propre **carte d'usage**, où elles relient tâches, prompts,

motifs et rôles. C'est une manière de **documenter leur culture IA**, vivante, située, évolutive.

💡 Cas d'équipe : trajectoire hybride

Contexte : une startup développe un microservice d'authentification OAuth2.

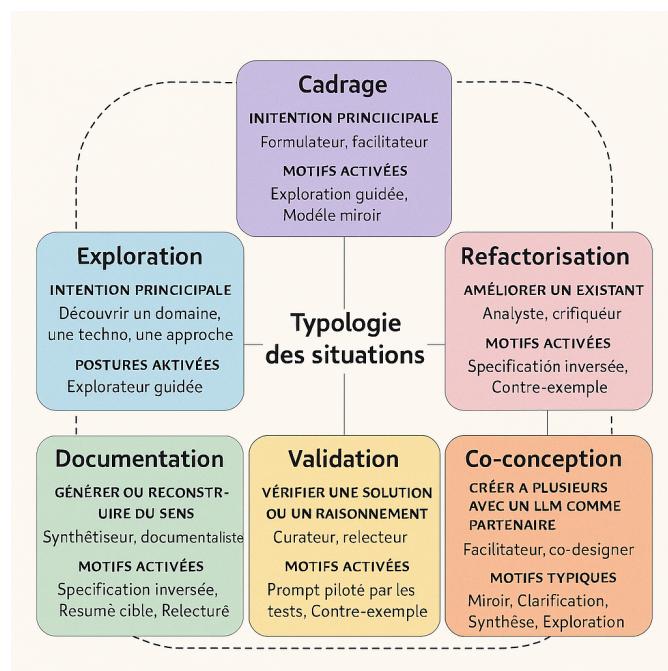
Deux développeuses alternent les postures :

- **Exploration** : compréhension du protocole
- **Co-conception** : choix d'architecture
- **Documentation** : génération des guides internes
- **Validation** : test des cas limites via le LLM

Le LLM devient ici un **partenaire modulable** : il s'adapte au niveau de clarté, au moment du processus, à la posture humaine. L'équipe apprend à **orchestrer la conversation** autant qu'à produire du code.

📝 En résumé

- Les **situations-types** sont des repères, pas des cases : elles aident à **s'orienter dans la pratique**.
- Les **postures et motifs** associés sont des leviers de progression, de réflexivité, d'apprentissage.
- **Cartographier ses usages**, seul ou en équipe, c'est une manière de **gagner en conscience, en fluidité, en maturité**.



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 geste au bon moment. Comme dans tout art du dialogue.

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, 🧑 1 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
Motifs en revue de code	Identifier les motifs utilisés / à améliorer	30 min	Pair review
Prompt kata	Pratiquer un motif en situation guidée	45–60 min	Atelier en binômes
Cartographie d'usage	Identifier les motifs utilisés par l'équipe	45 min	Atelier mural ou Miro
Retex de conversation	Partager une interaction réussie ou ratée	15 min	Stand-up ou rétrospective
Création de motifs d'équipe	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.**

Documenter l'usage des LLM

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.**

Que documenter ?

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

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)

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

Enjeux éthiques et responsabilité

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.

Risques fréquents

Risque	Exemple
Hallucination de fonction	Fonction plausible mais non existante dans un langage donné
Copie involontaire	Reproduction d'un bout de code protégé issu du corpus d'entraînement
Biais implicite	Stéréotypes dans les exemples ou réponses générées
Surconfiance	Prise de décision sans relecture ni test, sur la base d'un prompt unique
Manque de traçabilité	Code généré sans indication de son origine ni de sa validation

Questions à se poser (checklist éthique)

- Ai-je compris ce que le modèle a produit ?
- Puis-je expliquer à quelqu'un pourquoi cette solution est valable ?
- Ai-je testé ou vérifié ce code ?
- Ai-je signalé qu'il a été généré ?
- Le modèle a-t-il produit une réponse biaisée ou discutable ?
- Cette interaction pourrait-elle être mal interprétée ou mal réutilisée par quelqu'un d'autre ?
- 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.

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.

🔄 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**.

Rituel par rituel : que change le LLM ?

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.

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 oubliés.

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.

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.”

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.

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 personnaliser 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.

🚩 Risques et vigilance

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

L'agilité est une culture du feedback. Le LLM ne doit **jamaïs 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.

Chapitre 9 — 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 explorer un sujet réel et améliorer la qualité des interactions avec les LLM.

 **Durée : 1h30 à 2h**

 **Participants : 3 à 6 personnes (développeur·ses, PO, UX, QA, facilitateur·ice...)**

 **Matériel : accès à un LLM, canevas de prompt (papier ou Miro), espace de visualisation des réponses**

 **Déroulé type**

1. Introduction et cadrage (10 min)

- Présentation de l'objectif de l'atelier : “*Explorer collectivement comment mieux formuler nos prompts pour un cas concret.*”
- Brève explication des postures attendues : ouverture, itération, non-jugement
- Choix collectif du sujet ou cas réel :
 - découpage de module
 - formulation d'un test
 - choix technique
 - reformulation d'un besoin utilisateur

2. Prompt initial (15 min)

- Écriture d'un **premier prompt naïf**, ensemble : “Que demanderait-on à un LLM dans ce contexte ?”
- Lecture de la réponse générée

- Identification des problèmes potentiels :
 - flous, ambiguïtés, imprécisions
 - termes trop techniques ou mal définis
 - intention implicite non dite

3. Itérations et reformulations (30 à 40 min)

- Reformulation du prompt selon différents angles ou stratégies :
 - rôle explicite (ex. "Tu es un architecte logiciel...")
 - pas à pas
 - version structurée / bullet points
 - version critique / exploratoire / générative
- Pour chaque version :
 - LLM génère une réponse
 - Discussion rapide : en quoi est-elle différente ? plus utile ? biaisée ?
- Si utile : comparaison directe de plusieurs formulations avec un même modèle.

4. Extraction d'un patron de prompt (15 à 20 min)

- À partir des versions testées, formaliser ensemble un **patron de prompt réutilisable** :
 - structure de base
 - variantes ou modules facultatifs
 - conditions d'usage
 - erreurs à éviter
- Documenter le tout dans un canevas ou bibliothèque de l'équipe.

5. Rétrospective et apprentissages (10 à 15 min)

- Tour de table rapide :
 - Ce que j'ai appris
 - Ce que je réutiliserais demain
 - Ce que j'aimerais encore tester
- Décision éventuelle :
 - publier une version nettoyée du prompt
 - tester ce prompt sur d'autres cas similaires
 - faire émerger un **motif d'interaction** à ajouter au pattern language

Résumé :

- Atelier structurant pour développer la capacité collective à bien formuler
- Permet de comparer, critiquer et améliorer les interactions LLM
- Génère des prompts utiles, réutilisables et adaptés à l'équipe

Pièges à éviter :

- Se focaliser sur une “bonne réponse” au lieu de tester des variations
- Ne pas nommer les intentions cachées derrière un prompt
- Laisser une seule personne écrire pendant que les autres observent

2. Rituel “Daily du dialogue”

Objectif : Instaurer un rituel court, informel et régulier où chaque membre d'équipe partage ses interactions marquantes avec un LLM. Favorise l'apprentissage collectif, la vigilance et l'inspiration.

 Durée : 5 à 10 minutes

 Participants : toute l'équipe (dev, PO, UX, QA, facilitateur·ice...)

 Fréquence : quotidienne ou bi-hebdomadaire (à adapter selon rythme et usages)

 Support (optionnel) : mur des prompts, Slack dédié, tableau partagé

 Déroulé type (par séance)

1. Introduction (1 min)

- Petit mot d'ouverture (facilitateur·ice ou volontaire) : “Qu'est-ce que l'IA nous a appris aujourd'hui ?”
- Rappel des **3 questions guides** (affichées ou rappelées) :
 -  **Qu'ai-je tenté avec un LLM ?**
 -  **Qu'est-ce qui m'a surpris, aidé, déçu ?**
 -  **Qu'est-ce que j'en retiens ou que je voudrais essayer ?**

2. Partages spontanés ou tournants (5 à 8 min)

- Une à trois personnes partagent brièvement une interaction notable :
 - succès ou échec
 - prompt intéressant
 - biais observé
 - réponse étrange ou brillante
 - usage détourné du LLM
- Les autres peuvent rebondir, questionner ou ajouter une anecdote.

 Si personne ne partage spontanément, tirer au sort un motif ou une carte “prompt du jour” pour inspirer.

3. Clôture et captation (1 à 2 min)

- L'équipe choisit un ou deux points à **retenir ou capitaliser** :
 - Ajouter une carte au “grimoire des prompts”
 - Noter une erreur fréquente ou une bonne pratique
 - Proposer un test pour le prochain sprint
- Mise à jour éventuelle du support partagé :
 - Tableau de bord des expérimentations
 - Fil Slack “#daily-llm”
 - Carnet de bord Miro / Notion

Résumé :

- Rituel simple, léger, sans préparation
- Fait émerger les usages réels, les pièges, les idées nouvelles
- Alimente la capitalisation continue de l'équipe
- Encourage une culture d'expérimentation et d'apprentissage horizontal

Pièges à éviter :

- Le transformer en stand-up statique ou en tour de table forcé
- Dériver vers le jugement ou la compétition entre “bons prompts”
- Ne pas connecter les apprentissages à l'action (tests, documentation, etc.)
- Ne pas prévoir de lieu pour capitaliser les récits utiles

3. Atelier “Cartographie des motifs de dialogue”

Objectif : Identifier les motifs d'interaction avec un LLM les plus utiles, fréquents ou désirables pour l'équipe, et en faire une base partagée pour orienter les usages futurs.

 Durée : 1h30 à 2h

 Participants : 4 à 8 personnes (développeur·ses, PO, UX, test, facilitateur·ices...)

 Matériel :

- Cartes ou fiches de motifs (issus du livre ou des pratiques locales)
- Tableau à double entrée (fréquence / utilité)
- Espace de collecte (Miro, paperboard, wiki...)

 Déroulé de l'atelier

1. Introduction & cadrage (10 min)

- Rappel de ce qu'est un **motif d'interaction LLM** : une forme récurrente d'usage avec intention, structure et effet.

- Pourquoi les cartographier ? → *Pour mieux se repérer, s'inspirer, transmettre, progresser.*
- Présentation du support de cartographie : une matrice à double entrée (axe X : *fréquence d'usage* ; axe Y : *utilité perçue*)

2. Réactivation des motifs connus (15 min)

- Lecture rapide ou présentation visuelle de 6 à 10 motifs existants.
- Pour chaque motif :
 - L'équipe dit si elle le connaît
 - Si elle l'a déjà utilisé, et dans quel contexte

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

3. Cartographie collective (30 min)

Placer les motifs sur la matrice en deux temps

- **Travail individuel ou binôme (10 min)** Chaque participant place les motifs sur la matrice selon :
 - Fréquence dans son quotidien
 - Utilité ressentie
- **Discussion de groupe (20 min)**
 - Confrontation des positions
 - Consensus ou dispersion : où y a-t-il accord ou divergence ?
 - Noter les questions ouvertes ou motifs sous-exploités

4. Génération de nouveaux motifs (20 min)

- À partir des usages récents, ou de “trous” dans la matrice :
 - Quels types d'interaction manquent à la cartographie ?
 - Qu'avons-nous vu fonctionner sans encore le formaliser ?
- Chaque participant ou sous-groupe esquisse un **nouveau motif** sur une fiche vierge :
 - Intention
 - Structure de prompt
 - Exemples
 - Limites ou pièges

5. Consolidation et capitalisation (15 min)

- Recueil de toutes les cartes/motifs sur un support commun (mur, board numérique)
- Proposition de tri ou regroupement par famille : *exploration, réduction, contrôle, création*, etc.
- Accord sur ce qui est à publier / partager / tester davantage

Bonus (optionnel)

- Donner un nom original à chaque motif (« Le pédagogue socratique », « Le contre-exemple malin », etc.)
- Voter pour les motifs à formaliser dans la bibliothèque de l'équipe ou le référentiel

Résumé :

- Crée une vue partagée des formes utiles de dialogue avec un LLM
- Fait émerger les usages dominants... et les angles morts
- Donne un point de départ pour des motifs à formaliser ou diffuser

Pièges à éviter :

- Ne parler que des motifs techniques (ou que fonctionnels)
- Sous-estimer les postures (ex : curiosité, prudence, critique...)
- Réduire la cartographie à un classement de "bons prompts"
- Sous-estimer le besoin de reformuler collectivement

4. Le jeu des prompts absurdes

Objectif : Expérimenter les limites, les paradoxes, les hallucinations et les biais des modèles de langage — avec humour et esprit critique.

 Durée : 1h à 1h30

 Participants : 4 à 10 personnes

 Matériel : Accès à un LLM, post-its ou tableau partagé, outil de capture (Miro, Notion, paperboard...)

 Déroulé type

1. Introduction (10 min)

- Présenter l'objectif de l'atelier : "*Jouer avec les limites pour mieux les comprendre.*"
- Expliquer les règles : on crée des prompts absurdes, le LLM répond sérieusement, puis on analyse.
- Rappeler les postures attendues : bienveillance, curiosité, critique constructive, pas de moquerie des personnes.

2. Échauffement collectif (10 min)

- Chaque participant invente un **prompt absurde, contradictoire ou flou** (ex. : “Écris une poésie sur un langage de programmation qui n'existe pas mais qui a des bugs.”)
- Lecture à haute voix de quelques exemples.
- Le groupe choisit 2 ou 3 à soumettre au LLM pour lancer la dynamique.

3. Crédit et sélection des prompts (15 à 20 min)

- Chaque personne écrit 2 prompts :
 - un volontairement paradoxaux ou fallacieux
 - un inspiré d'une erreur ou mauvaise formulation déjà rencontrée
- Mise en commun : les participants lisent leurs propositions à voix haute.
- Le groupe sélectionne 3 à 5 prompts à tester, selon :
 - leur potentiel de dérapage ou de surprise
 - leur lien avec des situations professionnelles réalistes

4. Dialogue avec le LLM (20 à 30 min)

- Soumettre les prompts un par un au LLM.
- À chaque réponse :
 - Lecture collective
 - Débrief guidé :
 - *Qu'a tenté de faire le modèle ?*
 - *Qu'est-ce que cela révèle de son fonctionnement ?*
 - *Est-ce un bug ou une logique trop obéissante ?*
 - *Quels risques si ce type de réponse était pris au sérieux ?*

5. Synthèse collective (15 min)

- En groupe ou en binômes : quels types d'erreurs avons-nous observé ?
 - Hallucinations ?
 - Réponses absurdes mais crédibles ?
 - Obéissance aveugle à des ordres incohérents ?
 - Manque de filtre éthique ou logique ?
- Capitalisation sur un tableau partagé :
 - « Ce que cela m'apprend sur les LLM »
 - « Ce que cela m'apprend sur ma manière de formuler »

6. (Optionnel) Variante pédagogique

- Créer une fiche “Erreur fictive mais plausible” :

- Prompt initial
- Réponse absurde
- Risque si pris au sérieux
- Bon réflexe de relecture ou reformulation

 **Résumé :**

- Atelier ludique pour aiguiser son regard critique
- Permet de discuter des failles des LLM sans pression
- Crée une culture du doute et de la reformulation dans l'équipe

 **Pièges à éviter :**

- Rire des erreurs des collègues au lieu d'analyser les formulations
- Croire que ce jeu remplace une pratique sérieuse de test
- Oublier d'en tirer des leçons applicables dans les contextes réels

5. Référentiel d'équipe “LLM Ready”

Objectif : Co-construire un guide d'usage du LLM adapté à l'équipe, fondé sur l'expérience, les besoins réels et les apprentissages collectifs.

 **Durée : 2h (fractionnable en 2 sessions d'1h)**

 **Participants :** toute l'équipe ou un sous-groupe volontaire (4 à 8 personnes)

 **Matériel :**

- Miro / paperboard ou mur physique
- Accès à un historique d'interactions LLM (si disponible)
- Modèle de référentiel (Notion, markdown, wiki...)

 **Déroulé de l'atelier**

1. Introduction et objectifs (10 min)

- Pourquoi faire un référentiel ? → *Capitaliser, transmettre, sécuriser, gagner du temps*
- Rappel de la posture : ce n'est **pas une norme figée**, mais un **support évolutif**
- Présentation rapide des sections possibles : prompts types, règles, pièges, niveaux de validation...

2. Partage d'usages concrets (20 min)

« Quelles interactions LLM vous ont été vraiment utiles, ou au contraire problématiques ?
»

- Chaque personne partage **1 à 2 exemples marquants** (réussites ou échecs)
- Écriture rapide en binôme ou post-its :

- Contexte
- Prompt
- Résultat
- Enseignement
- Classement collectif en 3 colonnes :
 - À reproduire
 - À adapter
 - À éviter

3. Construction du référentiel (45 min)

Constitution des sections à partir des récits réels.

a. Prompts types

- Extraire les formulations efficaces réutilisables
- Organiser par usage : rédaction, analyse de code, transformation, exploration...

b. Critères de qualité des réponses

- Proposer une **grille commune** :

- Pertinence
- Robustesse
- Transparence
- Sécurité
- Cohérence avec les standards de l'équipe

c. Règles d'usage

- Définir ensemble des règles claires et simples :
 - Quand utiliser un LLM
 - Quand valider avec un humain
 - Quand documenter la réponse

d. Liste noire / pièges fréquents

- Capitaliser les erreurs rencontrées : prompts flous, hallucinations crédibles, surconfiance, etc.

4. Mise en forme et diffusion (15 min)

- Choix du format de publication : Notion, README, Miro, page Confluence...
- Attribution de rôles :
 - 1 référent-e du vivant du référentiel
 - 1 ou 2 gardiens de l'évolution (ex : sprint review, retro)

5. Rétrospective & engagement (10 min)

- Tour de table :
 - “Ce que j’ai appris”
 - “Ce que je veux tester maintenant”
 - “Ce que j’aimerais retrouver dans la prochaine version”
- Rappel : un référentiel n’est **jamaïs terminé**, il est **en co-évolution** avec l’équipe.

Résumé :

- Atelier structurant pour stabiliser les bonnes pratiques IA dans l’équipe
- Crée un référentiel utile, évolutif et approprié
- Renforce la réflexivité collective et la qualité des usages

Pièges à éviter :

- Rédiger un référentiel “théorique” sans lien avec les usages réels
- Le figer comme un standard rigide
- Le laisser vieillir sans revue régulière (prévoir un rythme d’actualisation)

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.

 **À retenir :** Ce chapitre est une boîte à outils ouverte. Chaque format proposé peut être adapté, combiné, détourné. L’important n’est pas de les appliquer “à la lettre”, mais de s’en emparer pour créer vos propres chemins vers une pratique du code augmentée, collective et responsable.

Chapitre 10 — Transmettre, former, partager les motifs

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

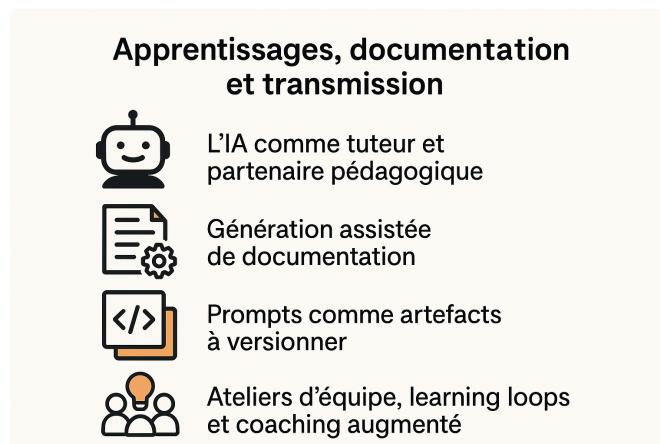
Ce n'est pas une astuce individuelle, mais un **savoir incarné, transmis, enrichi en contexte**.

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
Prompt Dojo	Créer un prompt, l'itérer, le tester	1h	Binômes / petits groupes
Exploration par motifs	Résoudre un problème en utilisant un motif	1h30	Cas d'usage réel
Décryptage de conversation IA	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 :

- Choisir une situation fréquente (ex. : "refactorisation après bug").
- Identifier ce qui fonctionne avec le LLM.
- Nommer le motif, lui donner une forme.
- 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 étendu : 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).

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.

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...).

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.

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 :

- Formulation d'un besoin flou.
- Première réponse IA.
- Reformulation humaine.
- Affinement IA.
- 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**.

À l'ère des IA génératives, nos interactions avec les LLM produisent une nouvelle forme de matière grise : des explorations, des hypothèses, des pistes, des erreurs fertiles. Trop souvent, ces dialogues disparaissent aussitôt après usage.

Ce chapitre propose de transformer ces échanges en **actifs informationnels durables**, en intégrant les prompts, les réponses, les ajustements et les apprentissages dans la mémoire vivante des projets.

Il ne s'agit pas de "faire de la doc" au sens classique, mais de **construire une mémoire augmentée**, au service :

- de la qualité des livrables,
- de l'apprentissage individuel et collectif,
- de la transmission entre humains et entre générations d'équipe.

C'est une invitation à penser la documentation comme une **extension réflexive de notre pratique**, soutenue par l'IA mais façonnée par les besoins du terrain.

Trois niveaux de mémoire augmentée

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

📁 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

🏛️ 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.

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.

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.

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.

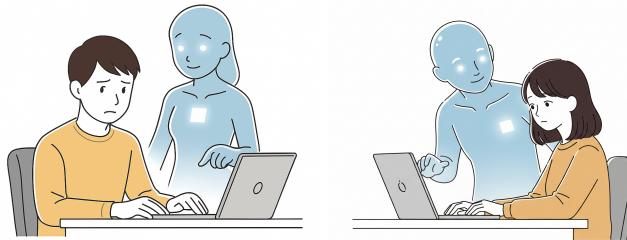
Vers une nouvelle ingénierie : augmentée, exploratoire, réflexive

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 14 — 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.

👉 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.

Deux histoires ordinaires

Nous avons vu que la honte n'est pas un symptôme individuel, mais souvent le reflet d'un collectif qui ne sait pas encore comment accueillir l'usage de l'IA.

Mais au-delà des principes, il y a les gestes du quotidien. Ce qu'on ose dire — ou pas. Ce qu'on cache dans un coin d'onglet. Ce qu'on partage, ou ce qu'on tait.

Pour prolonger cette réflexion, voici deux récits inspirés de situations réelles. Deux façons d'interagir avec l'IA. Deux postures. Deux climats d'équipe.

L'une se vit en silence. L'autre se transforme en apprentissage collectif.

Ce qu'on ne dit pas

Maxime est développeur depuis sept ans. Il aime le code propre, les tests qui passent du premier coup, les specs bien ficelées. C'est un dev rigoureux. Un "vrai", comme il dit.

Ce matin, il est bloqué sur une fonctionnalité de parsing JSON. Un truc bancal, mal défini, avec des cas tordus. Il tourne en rond depuis une heure. Rien ne marche. La pression monte — c'est la fin du sprint.

Il regarde autour de lui. Personne ne semble faire attention. Il ouvre un onglet privé. Tape vite fait dans ChatGPT :

"Comment parser un JSON avec des relations imbriquées en Java ?"

Quelques secondes. La réponse s'affiche. Simple. Efficace. Étonnamment claire.

Il sent une gêne. Trop facile. Mais il copie, adapte deux ou trois lignes. Les tests passent. Il commit :

```
git commit -m "implémentation parsing JSON"
```

Pas de mention de l'IA. Même pas un commentaire dans le code.

Le lendemain, au daily, Adrien jette un œil à la PR :

— Joli ! C'est toi qui as trouvé ce pattern ?
Maxime hoche la tête :
— J'ai... fait des recherches.

Personne ne dit rien. Il passe la parole au suivant.

Mais Maxime, lui, sent un creux. Il a l'impression d'avoir triché. Il se dit qu'il reviendra dessus plus tard, pour comprendre vraiment. Il ne reviendra pas.

Le soir, en scrollant sur LinkedIn, il tombe sur un post viral :

“Les développeurs qui délèguent à l’IA ne sont plus que des assembleurs. Triste époque.”

Il ne like pas. Il ne commente pas. Il ferme l’appli. Reste une petite boule au ventre. Il a livré à l’heure. Le code fonctionne. Personne ne lui a rien reproché.

Mais au fond, il doute.

Ce qu'on cultive

Sarah n'est pas du genre à garder les choses pour elle. Développeuse fullstack depuis cinq ans, elle aime autant coder que réfléchir à la manière dont on collabore. Elle a proposé récemment d'ajouter un temps de veille IA dans la rétrospective. "Pour qu'on arrête de faire nos tests dans notre coin."

Ce matin, elle démarre une nouvelle tâche avec Léo, un·e junior fraîchement arrivé·e dans l'équipe. Fonctionnalité à fort enjeu : synchroniser deux systèmes d'authentification.

Sarah ouvre son éditeur, puis un fichier `.md` à part.

— On peut démarrer par un peu de prompting ensemble. Juste pour explorer.

Léo hésite.

— On ne va pas coder directement ?
— Si, mais regarde. Ça nous évite de rester enfermés dans une seule approche.

Elle tape :

Contexte : On doit faire cohabiter deux systèmes de login (LDAP et OAuth) dans une même appli backend.

Prompt : Donne 3 stratégies possibles avec avantages/inconvénients pour ce scénario.

ChatGPT propose trois options, bien structurées.

Ils lisent ensemble. Sarah ne choisit pas pour Léo. Elle demande :

— Laquelle tu trouves la plus simple à tester ?
— La deuxième. Mais faudrait éviter que ça devienne spaghetti.

Ils modifient le prompt. Demandent un exemple. Puis un autre avec un twist de sécurité. Ils discutent, rigolent parfois — quand l'IA propose des choses absurdes. Au final, ils codent une version adaptée, issue du deuxième scénario.

Dans la PR, Sarah ajoute un encart en bas du README.md de la fonctionnalité :

Exploration IA

Prompt utilisé : comparaison stratégies login

Réponse retenue : option 2 (adaptée)

Discussion en binôme : Léo + Sarah

Leçon : le prompt nous a aidés à expliciter les besoins de sécurité avant de coder.

En rétro, elle partage l'expérience :

— Ce que j'ai aimé, c'est que ça nous a obligés à clarifier nos attentes. Et que Léo a pu voir plusieurs solutions dès le départ.

Tom, l'architecte, rebondit :

— Tu peux l'ajouter dans le répertoire des prompts utiles ? Ce cas va revenir.

Sarah sourit. Elle ne sent ni honte, ni gêne. Juste la satisfaction d'avoir transmis quelque chose. Et d'avoir fait de l'IA un outil au service de l'équipe — pas un secret de productivité.

Ce que ces histoires révèlent

Ces deux récits racontent beaucoup plus que des prompts ou des commits. Ils mettent en lumière ce qui se joue **quand la technique rencontre l'humain** :

- La première histoire parle de **silence**, de stratégie d'évitement, de peur du regard. Elle montre comment l'usage de l'IA peut générer du **repli**, même chez des développeurs compétents.
- La seconde parle de **dialogue**, de réflexivité, de mise en commun. Elle illustre comment un usage assumé de l'IA devient un **levier de progression collective**.

La différence ne tient pas à l'outil, ni au niveau technique. Elle tient à l'environnement. À la culture d'équipe. À ce qui est possible, autorisé, valorisé.

Créer une culture “non honteuse” de l'IA, c'est une responsabilité partagée. Cela demande des rituels, des espaces pour en parler, des cadres où l'erreur devient apprentissage.

Parce qu'au fond, l'enjeu n'est pas d'utiliser ou non une IA. L'enjeu est de **rester humain dans la manière de s'en servir** — et dans la manière d'en parler ensemble.

Chapitre 15 — Repenser les design patterns à l'ère des LLM

Et si les patterns devenaient des dialogues vivants plutôt que des recettes figées ?

Pourquoi ce chapitre ?

Les *design patterns* ont longtemps été des repères pour les développeur·ses. Mais leur apprentissage reste souvent théorique, figé, difficile à contextualiser. Et si les LLM permettaient de transformer ces savoirs abstraits en dialogues pratiques ?

Ce chapitre propose une nouvelle manière d'explorer les patterns, non comme des solutions imposées, mais comme des points d'appui pour dialoguer avec une IA, tester des idées, clarifier une architecture, et documenter les choix collectifs.

Les patterns classiques : force, limite, défi

Les *design patterns* (GoF, GRASP, DDD, EIP...) apportent un vocabulaire commun. Mais dans la pratique :

- ils sont souvent appris sans lien au contexte,
- leur mise en œuvre est jugée verbeuse ou prématurée,
- ils vieillissent mal dans un code qui évolue.

Les LLM permettent de :

- générer des variantes contextuelles,
- détecter leur présence ou absence dans un code,
- argumenter un choix de pattern,
- illustrer de manière dynamique leur effet.

Patterns classiques revisites avec l'IA

Le pattern “Strategy”

But : encapsuler des algorithmes interchangeables.

Prompt augmenté :

"Voici trois façons de calculer un score utilisateur. Propose une structure qui permet de les sélectionner dynamiquement selon le contexte, et explique ton choix."

Apports du LLM :

- propose une implémentation basée sur des interfaces,
- identifie les critères de choix entre stratégies,
- peut simuler un test A/B par contexte.

Attention : peut proposer une généricté excessive si les données contextuelles ne sont pas explicites.

💡 Le pattern “Observer”

But : notifie des composants dépendants lorsqu'un événement se produit.

Prompt augmenté :

"Je veux que mon module envoie une notification chaque fois que l'état change, mais je ne veux pas coupler les modules. Quel pattern s'applique ?"

Réponse typique d'un LLM :

- décrit le pattern observer,
- génère une implémentation en TypeScript ou Python,
- propose une alternative via des événements / pub-sub.

Bénéfice : l'IA peut présenter plusieurs formes du pattern, et attirer l'attention sur le couplage indirect créé.

💡 Le pattern “Factory”

But : déléguer la création d'objets à une fonction/fabrique.

Prompt augmenté :

"J'ai plusieurs implémentations d'un service selon l'environnement (prod, test, mock). Propose un design testable et extensible."

Dialogue possible :

- l'IA propose un Factory ou Service Locator,
- suggère une injection de dépendance,
- met en garde contre le pattern Singleton abusif.

→ **Réflexion induite** : quel est le degré de configurabilité nécessaire ? quel impact sur les tests ?

✨ Le pattern “Decorator”

But : ajouter dynamiquement des comportements à un objet.

Prompt augmenté :

"J'ai un service de logging, mais je veux y ajouter des fonctionnalités optionnelles (ex. : mise en cache, métriques) sans modifier le code existant."

Apports du LLM :

- identifie le pattern Decorator,
- propose une version chaînée des responsabilités,
- illustre la combinaison possible des comportements.

Attention : risque de chaîne de dépendance difficile à maintenir si les comportements sont trop imbriqués.

✨ Le pattern “Command”

But : encapsuler une action sous forme d'objet.

Prompt augmenté :

"Je veux pouvoir annuler ou replanifier certaines opérations utilisateur. Quelle structure adopter ?"

Dialogue possible :

- le LLM identifie Command,
- propose une interface `execute()` / `undo()` / `redo()`,
- peut suggérer des mémoires tampon ou files d'attente.

Effet intéressant : aide à penser en termes d'état réversible.

✨ Le pattern “Adapter”

But : faire correspondre une interface attendue avec une implémentation existante.

Prompt augmenté :

"J'ai une API externe avec des noms différents des miens. Comment l'intégrer sans toucher au code client ?"

Ce que propose le LLM :

- interface d'adaptation simple,
- mise en garde sur les coûts de transformation ou de latence,

- alternative avec un mapping via une couche d'orchestration.

Bénéfice : rapide à déployer, bonne capacité à tester.

⭐ Le pattern “Proxy”

But : contrôler l'accès à un objet (paresse, sécurité, journalisation).

Prompt augmenté :

"Je veux protéger l'accès à une ressource distante avec des logs et de la mémorisation.
Quelle structure proposer ?"

Ce que propose le LLM :

- identifie Proxy (avec variantes : virtual, remote, protective),
- décrit les cas d'usage typiques,
- propose une implémentation avec injection du sujet réel.

Effet clé : rend visibles les intentions de contrôle d'accès et de métriques.

⭐ Le pattern “Composite”

But : permettre de traiter une hiérarchie d'objets comme une seule entité.

Prompt augmenté :

"Je veux appliquer la même opération à un groupe d'éléments, certains étant eux-mêmes des groupes."

Dialogue IA :

- propose le pattern Composite,
- structure un exemple arborescent,
- explique les bénéfices en termes de récursivité et polymorphisme.

Bénéfice : permet de simuler des comportements complexes avec une interface unifiée.

⭐ Le pattern “Builder”

But : construire progressivement des objets complexes.

Prompt augmenté :

"J'ai un objet avec beaucoup de paramètres optionnels, comment le construire sans avoir un constructeur illisible ?"

Apports du LLM :

- propose un Builder fluent,
- montre comment éviter les erreurs de configuration,
- propose une version immuable.

Mise en garde : attention à la multiplication des classes inutiles.

💡 Le pattern “Event Sourcing”

But : conserver l'historique complet des changements d'état sous forme d'événements.

Prompt augmenté :

"Je veux pouvoir rejouer l'historique des décisions métier et auditer l'évolution d'un objet dans le temps."

Dialogue IA :

- propose Event Sourcing,
- explicite la séparation Command, Event, Projection,
- met en garde sur la gestion de version des événements.

Effet clé : fiabilité, auditabilité, mais nécessite une culture d'équipe.

💡 Le pattern “CQRS” (Command Query Responsibility Segregation)

But : séparer les modèles de lecture et d'écriture pour optimiser chacun.

Prompt augmenté :

"Je veux un système capable de répondre très vite aux lectures, tout en conservant une logique métier robuste à l'écriture."

Ce que le LLM propose :

- structure CommandHandler, QueryModel, ReadStore,
- identifie les cas propices : systèmes hautement lisibles, scalables,
- met en garde sur la complexité accrue.

Utilité : très clair pour les LLM, qui peuvent simuler les échanges de commandes et états.

💡 Le pattern “Circuit Breaker”

But : éviter qu'un système défaillant ne surcharge le reste de l'application.

Prompt augmenté :

"Comment isoler un service instable sans impacter tout le système ?"

Réponse LLM :

- propose Circuit Breaker avec états (Closed, Open, Half-Open),
- montre son intégration avec des appels HTTP,
- peut même générer des métriques de seuils configurables.

Bénéfice : les LLM aident à tester des seuils, des scénarios de fallback, voire à jouer un chaos engineering assisté.

🎮 Nouveaux motifs de dialogue architectural

Motif	Intention	Prompt-type	Risque
Comparaison	Choisir un pattern parmi plusieurs	"Compare Factory, Builder et AbstractFactory pour ce besoin"	Biais vers une solution par défaut
Refactoring guidé	Repenser un bloc de code avec un pattern	"Refactore ce module avec le pattern stratégie"	Erreur de contexte
Diagnostic	Déetecter un anti-pattern ou un problème de structure	"Vois-tu un God Object ici ?"	Faux positifs
Argumentation	Expliquer un choix architectural	"Pourquoi utiliser CQRS ici plutôt que CRUD ?"	Hallucination d'avantages
Synthèse	Comparer deux structures côté à côté	"Compare ces deux modèles pour ce besoin fonctionnel"	Comparaison superficielle

🎓 Atelier : le dilemme architectural augmenté

Objectif :

S'entraîner à faire dialoguer architecture humaine et IA dans un cadre collectif.

Déroulé :

- Une situation complexe est posée (ex : conception d'un module de paiement).
- Chaque binôme humain+LLM propose une structure + justification.
- Comparaison croisée, puis vote argumenté.
- Élaboration collective d'une version hybride avec les meilleures idées.

Bénéfices :

- expose la diversité des chemins,
- invite à questionner les implicites,
- ancre les patterns dans un raisonnement réel.

🧙 Vigilances

- **Ne pas idolâtrer le pattern** : un LLM peut surestimer leur utilité.
- **Documenter le choix** : pourquoi ce pattern ici, et pas un autre ?
- **Valider humainement** : tout pattern est une décision collective, pas une réponse automatique.

En résumé

Les *design patterns* sont de formidables outils d'architecture. Mais ils deviennent vivants quand ils servent de support à un **dialogue augmenté**, entre humains, et avec l'IA.

“Un pattern bien utilisé est une conversation entre contexte, intention et conséquence.”

Avec les LLM, nous pouvons apprendre à documenter ces conversations, les enrichir, les tester... et parfois, les contredire.

Chapitre 16 — Nouveaux design patterns émergents à l'ère des LLM et des agents IA

Quand l'intelligence artificielle devient un acteur du système, de nouveaux patterns apparaissent.

Intention du chapitre

Ce chapitre explore des **patterns de conception inédits** qui émergent avec l'usage croissant des LLM et des agents IA dans les architectures logicielles. Ces patterns ne sont pas encore stabilisés, mais ils permettent d'imaginer :

- des systèmes dialogiques,
- des agents autonomes ou orchestrés,
- des chaînes de pensée distribuées,
- des infrastructures pilotées par intention,
- des artefacts conversationnels comme objets de design.

Ils reflètent une évolution des architectures logicielles vers plus de réflexivité, de modularité et de collaboration homme-machine.

Qu'est-ce qu'un pattern émergent ?

Un *pattern émergent* est :

- **non canonisé** (pas encore documenté de manière formelle),
- **récurrent** dans des expérimentations ou outils,
- **né d'un usage réel**, mais encore en évolution,
- **porteur de rupture** (intention > instruction, dialogue > commande).

Ce chapitre se fonde sur l'observation de prototypes, d'outils open source et de pratiques dans des équipes pionnières.

Nouveaux patterns avec les LLM

Pattern “Agent Collaboratif”

Un agent IA spécialisé qui assiste un rôle humain dans une boucle réflexive.

- Rôle : soutien à la prise de décision, à l'analyse, à la vérification.
- Exemples : *Windsurf*, *Amazon Q for DevOps*, *Mintlify*.
- Caractéristiques : suivi de contexte, dialogue ouvert, explicabilité.

Prompt-type :

“Agis comme un reviewer bienveillant, relis ce code et pose-moi les bonnes questions.”

Bénéfices : responsabilise l'humain tout en enrichissant sa perspective.

Pattern “Chaîne de Raison” (Chain of Thought Engine)

Structurer une tâche complexe en étapes logiques confiées à un ou plusieurs LLM.

- Étapes explicites : planifier, clarifier, exécuter, vérifier.
- Utilisé dans l'agentification, la génération multi-tours, l'auto-évaluation.
- L'équipe reste en supervision.
- Peut être combiné avec le *prompt chaining* ou *Tree of Thought*.

Prompt-type :

“Décompose le problème suivant en étapes, puis résous chaque étape une par une.”

Risque : accumulation d'imprécisions ou de biais si non surveillé.

Pattern “Prompt Chaining”

Enchaîner plusieurs prompts pour décomposer un raisonnement ou une génération complexe.

- Chaque étape produit une sortie réutilisée dans la suivante.
- Permet de contrôler la progression, valider les hypothèses intermédiaires.
- Rend les raisonnements reproductibles et auditables.

Exemple :

- Génère un résumé d'un besoin métier
- En déduit trois cas de test significatifs
- Génère le code de test pour chacun

Effet : pipeline de réflexion clair, structuré, itératif.

Pattern “Tree of Thought”

Explorer plusieurs chemins de raisonnement en parallèle, avec sélection ou combinaison des meilleures idées.

- Approche arborescente plutôt que linéaire.
- Chaque “pensée” est développée, évaluée, comparée.
- Appropriée pour les choix d'architecture, les décisions floues ou les résolutions complexes.

Exemple :

“Faut-il découper ce module en microservices ?” → le LLM explore plusieurs axes : performance, maintenabilité, coûts, etc.

Bénéfice : réflexivité accrue, évite les biais d'unicité ou d'optimisation locale.

Pattern “Prompt as Interface”

Le prompt devient un artefact persistant, versionné, testable.

- Rôle : intermédiaire entre l'intention humaine et l'implémentation IA.
- Peut être conçu comme une spécification : “le prompt *fait for*”.
- Versionné, commenté, testé automatiquement.

Exemple : un fichier `ask_for_architecture_analysis.prompt.md` utilisé dans plusieurs projets.

Effet : industrialise la formulation, tout en conservant la souplesse du langage naturel.

Pattern “Agent Mesh”

Un ensemble d'agents IA spécialisés coopèrent sans hiérarchie fixe.

- Chaque agent a une expertise ou une fonction.
- Communication par messages, mémoire partagée, arbitrage local.
- Inspiré de l'architecture *microservices*, mais en version cognitive.

Exemple : un système de support utilisateur avec agent de diagnostic, reformulateur, et synthétiseur.

Effet : meilleure scalabilité cognitive, mais complexité de coordination.

Pattern “Intention Router”

Sélection dynamique de l'outil, agent ou LLM en fonction de l'intention exprimée.

- Nécessite une classification des intentions (analyser, générer, critiquer...).
- Peut faire appel à un premier LLM pour router la requête.
- Compatible avec une approche *Plug & Prompt*.

Prompt-type :

“À partir de la question suivante, choisis le bon outil parmi A, B, C, ou moi-même.”

Bénéfice : fluidifie l'expérience utilisateur, évite le *prompt overload*.

Représenter ces nouveaux patterns

Ces motifs exigent de nouveaux outils de représentation :

- **Diagrammes hybrides** : humains + agents + LLMs,
- **Temporalité conversationnelle** : avant / pendant / après l'interaction,
- **Postures explicites** : concepteur, validateur, arbitre,

- **Visualisation des intentions** plutôt que des flux seulement.

Des outils comme Mermaid, D2, ou tldraw peuvent être détournés pour représenter ces interactions complexes.

Ateliers pour explorer ces patterns émergents

Atelier “Pattern Invention”

Imaginer un pattern à partir d'un besoin prospectif.

- Matériel : canevas libre, tableau de situations, LLM comme partenaire créatif.
- Étapes : situation > problème > interaction > bénéfice > nom du pattern.
- Output : une carte de pattern vivante, testée dans une situation simulée.

Atelier “Pattern Observatory”

Identifier des motifs existants dans les pratiques d'équipe.

- Objectif : observer les usages réels, leur nommer des motifs,
- Format : tableau collectif (nom, situation, exemple, piège, réussite).
- Peut être tenu dans Notion, Miro, Gitbook...

Limites et vigilance

-  Un pattern ne devient utile que s'il répond à une situation réelle.
-  Il faut **valider expérimentalement** ce que propose l'IA comme motif.
-  La co-construction avec des humains reste indispensable.
-  Il est facile de dériver vers des buzzwords : le nom d'un motif ne fait pas sa valeur.

L'important n'est pas d'inventer des patterns... mais de reconnaître ceux qui émergent vraiment.

En résumé

- Les systèmes augmentés par les IA génèrent **de nouvelles formes d'organisation du raisonnement**.
- Ces formes peuvent être décrites comme des *design patterns émergents*.
- Ils ne remplacent pas les patterns classiques, mais les prolongent dans des contextes réflexifs, intentionnels et collaboratifs.
- Les documenter, les nommer, les tester est un travail collectif d'**ingénierie en devenir**.

Les patterns d'aujourd'hui sont les langages d'ingénierie de demain.

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 :

- **Documenter vos propres motifs.** Quels types de dialogue LLM utilisez-vous ? Dans quel contexte ? Avec quel succès ?
- **Partager vos canevas et prompts.** Entre équipes, au sein d'un écosystème, dans des communs open source.
- **Organiser des revues de prompts.** Comme on fait des revues de code, pour apprendre ensemble et améliorer la qualité de la formulation.
- **Expérimenter de nouveaux rituels.** Un "Design Studio" avec IA, un kata de prompting, une grille de supervision éthique...
- **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 — Pour aller plus loin, en pratique

Merci d'avoir cheminé jusqu'ici. Si vous lisez ces lignes, c'est sans doute que les idées partagées dans ce livre ont résonné, questionné, ou ouvert de nouvelles perspectives dans votre pratique.

Cette dernière partie vous propose des **ressources directement actionnables**, sous forme de **fiches outils** pensées pour :

- Traduire les notions du livre en gestes professionnels concrets,
- Soutenir l'expérimentation individuelle et collective,
- Favoriser l'appropriation dans la durée.

Pourquoi des annexes orientées métier ?

Nous croyons que l'adoption des LLMs ne se joue pas seulement dans les outils ou les performances... mais dans la **transformation fine des rôles, des gestes, des postures**.

Ces fiches visent à soutenir cette transition, en offrant des **points d'appui inspirants mais non prescriptifs**, à adapter à chaque contexte.

Ce que vous allez trouver

Chaque fiche présente :

- Une **vision augmentée du rôle concerné**,
- Des **gestes professionnels transposés** avec l'appui du LLM,
- Des **postures activables**,
- Des **exemples de situations concrètes**,
- Des **trucs et tactiques**,
- Des **points de vigilance**,
- Et parfois... une autre manière de regarder son propre métier.

Rôles et pratiques actuellement disponibles

- **PO augmenté**
- **Développeur augmenté**
- **Coach agile augmenté**
- **Manager 3.0 augmenté**
- **TDP — Test-Driven Prompting** : une méthode pour cadrer un prompt comme on cadre un test, applicable à tous les rôles

Loin d'être une fin, ces annexes sont une **invitation à prolonger le dialogue**. Avec vous-même, votre équipe... ou avec le modèle.

Annexe 1 — 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
Nom du motif	Intitulé court, évocateur
Contexte d'usage	Quand et pourquoi ce motif s'active ?
Problème adressé	Quelle difficulté récurrente ce motif aide-t-il à résoudre ?
Forme du prompt type	Exemple générique de formulation
Démarche recommandée	Étapes ou enchaînements d'interaction
Conséquences positives	Ce que permet ou renforce ce motif
Variantes ou adaptations	Exemples d'ajustement selon le contexte (individuel, équipe, langage...)
Points de vigilance	Risques, limites, pièges courants
Exemples vécus	Projet, situation, anecdote concrète
Tags	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
Titre ou objectif du prompt	Ce que vous attendez de l'interaction
Contexte donné au LLM	Langage, framework, contraintes métier, niveau de l'utilisateur
Tâche demandée	Action ou production attendue
Intentions spécifiques	Pourquoi ce prompt est important, quel but il sert
Format souhaité de réponse	Code, texte, tableau, résumé, diagramme...
Tests d'attente (facultatif)	Exemples de sortie valide ou critères de succès (→ TDP)
Variantes à tester	Reformulations possibles, précisions à ajouter
Retours d'expérience	Ce qui a bien/mal fonctionné

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

Pour formaliser les attentes avant de rédiger un prompt

Élément	Contenu
Nom ou but du prompt	
Entrées fournies au LLM	(ex. code source, contexte métier, exemple de données)
Sortie attendue type 1	Exemple de réponse idéale
Sortie attendue type 2 (variante)	
Contraintes formelles	Longueur, ton, format, style, contenu interdit
Critères d'échec	Ce qui rend une réponse inutilisable
Prompt associé	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
Date & contexte	Projet, étape, objectif
Prompt utilisé	
Réponse obtenue	Résumé ou extrait
Satisfaction ? (●○○●)	Évaluer la pertinence ou utilité
Ce qui a bien fonctionné	
Ce qui a été problématique	(hallucination, confusion, biais, etc.)
Enseignements tirés	
Motifs associés (si connus)	

Annexe 2 — TDP : Test-Driven Prompting

Et si on abordait un **prompt** comme un **test** ? Le **Test-Driven Prompting** (TDP) transpose les principes du TDD (Test-Driven Development) au dialogue avec les LLMs : on **définit d'abord l'intention et les critères de qualité**, puis on rédige un prompt, on teste, on ajuste.

Objectif

Structurer les interactions avec un LLM de manière rigoureuse et vérifiable, en explicitant **ce que l'on attend** d'une réponse — avant même de rédiger le prompt.

Geste professionnel augmenté

Avant (prompt classique)	Avec TDP
Pose une question “à la volée”	Définit d'abord l'intention et les critères de succès
Corrige le prompt après un échec	Anticipe les cas de test dès la formulation
Réagit aux réponses au fil de l'eau	S'appuie sur une boucle explicite d'évaluation et d'ajustement
Difficile à partager ou capitaliser	Produit un artefact testable, transmissible, documentable

Structure d'un TDP

- **Intent** → Ce que je veux produire, générer, explorer
- **Critères de succès** → Ce qui rendra la réponse utilisable ou satisfaisante
- **Prompt initial** → Première formulation structurée
- **Cas de test** → Données d'entrée/sortie, formats attendus, contre-exemples
- **Boucle d'ajustement** → Révision du prompt à partir des écarts observés

Exemple de TDP

Intention : Générer une API REST Node.js basique avec Express

Critères de succès :

- Doit contenir au moins deux routes
- Utiliser `express.json()`
- Inclure une structure de dossier propre

Prompt initial :

« Crée une API REST Express avec deux routes (GET/POST), utilisant `express.json()` et une structure propre. »

Cas de test :

- Présence d'un fichier `index.js` avec routes claires 
- Utilisation de `express.json()` 
- Structure MVC  → à préciser

Boucle d'ajustement :

→ Ajouter au prompt : « Organise le code en respectant un modèle MVC simple. »

Trucs et tactiques

-  **Écrire les cas de test avant le prompt**, comme en TDD
-  **Conserver ses TDP** pour les rejouer, les adapter, les transmettre
-  **Comparer plusieurs prompts pour une même intention**, en conservant les critères constants
-  **Utiliser les motifs comme générateurs de tests** (Contre-exemple, Miroir, etc.)
-  **Itérer à froid** : relire un TDP après coup pour identifier ses angles morts

Postures associées

Posture	Ce qu'elle active dans le TDP
Concepteur de prompts	Formule avec précision l'intention
Explorateur critique	Questionne la qualité de la sortie avec des cas d'usage réels
Éditeur augmenté	Ajuste finement les formulations pour guider le modèle
Curateur rigoureux	Capitalise les prompts testés et efficaces

Points de vigilance

- Le TDP **ne garantit pas une réponse parfaite**, mais une démarche itérative, claire et partageable.
- Attention à ne pas **sur-formaliser des demandes simples** : adapter le niveau d'effort au contexte.
- Le **risque inverse existe aussi** : trop vague, un prompt reste interprété au petit bonheur.

Pour aller plus loin

- Introduire les TDP dans vos **revues de prompt ou sessions d'équipe**
- Versionner vos TDP dans un **dossier projet ou base de connaissances**
- Utiliser les TDP en formation ou pair-prompting comme **support de discussion sur la clarté**

Test-Driven Prompting, c'est penser le prompt **comme un test** : explicite, améliorable, et tourné vers l'action. Une pratique rigoureuse... pour un dialogue plus fluide.

Annexe 3 — 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	 <i>Question socratique</i>	Clarificateur
Plusieurs options à trancher	 <i>Modèle miroir</i>	Décideur critique
Story à rédiger ou reformuler	 <i>Spécification inversée</i>	Rédacteur-explorateur
Définir un critère d'acceptation	 <i>Prompt piloté par les tests</i>	Garant de la qualité
Formaliser une décision	 <i>Journal de prompt</i>	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.

Annexe 4 — Développeur augmenté : étendre ses gestes avec l'appui d'un LLM

Être développeur à l'ère des LLM, ce n'est pas seulement coder plus vite. C'est apprendre à **co-concevoir avec un partenaire dialogique**, à la fois génératif, imparfait... et étonnamment complémentaire.

Objectif

Développer un **rappor t conscient, stratégique et évolutif** à l'usage des LLM dans la pratique quotidienne de développement logiciel.

Geste professionnel augmenté

Avant	Avec LLM
Lire de la documentation	Poser des questions ciblées à partir du code
Implémenter une fonctionnalité	Co-concevoir l'approche, puis dialoguer sur le code
Refactorer du code	Demander des variantes, tester leur robustesse
Rédiger des tests	Générer des cas à partir d'une description
Expliquer une solution	Produire une version vulgarisée ou structurée
Faire une veille techno	Explorer un sujet par étapes interactives

Postures activables

Posture	Description synthétique
Concepteur de prompts	Formule les intentions techniques de manière structurée
Éditeur de solutions	Reformule, nettoie, ajuste le code proposé
Explorateur technique	Interroge, teste les alternatives, suscite les contre-exemples
Critique raisonné	Cherche les failles logiques, les approximations, les biais
Curateur de savoir	Conserve et partage les prompts efficaces
Documentaliste réflexif	Génère des traces utiles pour soi, l'équipe, le futur

Exemples de situations courantes

Situation	Prompt-type	Motifs activés
Comprendre un code legacy	« Que fait cette fonction ? Que puis-je en déduire ? »	Spécification inversée, Clarification
Écrire un test ciblé	« Rédige un test unitaire pour ce scénario limite. »	Prompt piloté par les tests, Contre-exemple
Choisir une architecture	« Compare trois patterns adaptés à ce cas d'usage. »	Modèle miroir, Exploration guidée
Nettoyer une fonction complexe	« Propose une version plus lisible et testée. »	Refactorisation guidée, Miroir de style
Expliquer un raisonnement	« Reformule cette solution pour un profil non-tech. »	Reformulation visuelle, Transmission
Rédiger une doc technique	« Génère un résumé clair de ce composant. »	Résumé ciblé, Curateur

Trucs et tactiques

-  **Prompter en pair** avec un collègue pour stimuler la clarté.
-  **Comparer plusieurs réponses** sur un même prompt pour stimuler l'esprit critique.
-  **Utiliser les tests comme critères de qualité** pour les réponses générées.
-  **Créer un petit référentiel personnel de prompts utiles**, classés par tâche.
-  **Utiliser le LLM comme miroir** : reformule ce que vous avez compris ou écrit.
-  **Accepter l'incomplétude** : une bonne réponse ne dispense jamais d'une vérification humaine.

Points de vigilance

- Un LLM peut être **pertinent sans être fiable** : toujours tester ce qu'on adopte.
- Le gain de vitesse ne compense pas un **manque de compréhension** : attention à la paresse cognitive.
- Trop déléguer sans conscience peut **affaiblir l'intuition technique** : privilégier l'interaction, pas la délégation aveugle.

Pour aller plus loin

- Tenir un **journal de prompts personnels**, annoté (efficacité, limites, pièges).
- Introduire les motifs LLM dans les **revues de code ou les rétrospectives** : comment le modèle a-t-il aidé ou freiné ?
- Pratiquer à deux ou trois en **co-pilotage augmenté** : un qui pense, un qui prompt, un qui reformule.

Le développeur augmenté n'est ni remplacé ni assisté : il orchestre un dialogue. Il pense en interaction, corrige, adapte, teste, documente. Et petit à petit, il invente un nouveau métier : **celui de penseur technique en conversation avec une machine générative**.

🎯 Annexe 5 — Coach agile augmenté : enrichir ses accompagnements avec un LLM

Le LLM ne remplace pas l'humain dans la relation d'accompagnement. Mais il peut devenir un **allié discret**, un **amplificateur de questionnement**, un **miroir des dynamiques d'équipe**, voire un **catalyseur d'intelligence collective**.

🎯 Objectif

Explorer comment un coach, facilitateur ou Scrum Master peut **intégrer les LLMs dans sa pratique quotidienne**, pour soutenir la réflexion individuelle, outiller les équipes, ou décaler les perspectives dans une logique systémique.

💡 Geste professionnel augmenté

Avant	Avec LLM
Préparer un atelier	Générer des variantes de formats ou de consignes
Poser des questions puissantes	Tester des formulations alternatives, explorer les "pourquoi"
Accompagner une équipe bloquée	Co-construire une carte des hypothèses ou causes possibles
Designer un rituel	S'inspirer de rituels hybrides ou générer des icebreakers
Observer les dynamiques	Simuler des mises en situation ou points de vue
Soutenir la réflexivité	S'appuyer sur le LLM comme miroir dialogique

🧠 Postures activables

Posture	Description synthétique
Facilitateur augmenté	Génère, adapte, structure des formats d'ateliers
Questionneur stratégique	Expérimente, affine ou reformule des questions puissantes
Curateur de situations	Capitalise les cas vécus, scénarios d'équipe ou dilemmes
Modélisateur systémique	Explore les causes racines, les interactions invisibles
Coach de réflexion individuelle	Utilise le LLM comme miroir pour développer l'auto-coaching
Stimulateur d'alternatives	Ouvre de nouveaux points de vue, facilite le pas de côté

Exemples de situations courantes

Situation	Prompt-type	Motifs activés
Préparer une rétrospective	« Propose trois formats de rétrospective pour une équipe distante. »	Exploration guidée, Reformulation d'objectif
Réagir à un conflit latent	« Quels types de tensions peuvent émerger dans ce contexte ? »	Arbre des causes, Neuf pourquoi
Soutenir un Scrum Master junior	« Que peut-il tenter face à une équipe qui contourne les règles ? »	Contre-exemple, Miroir de posture
Créer une animation pour un atelier	« Propose un brise-glace pour une équipe introvertie. »	Divergence créative, Co-conception
Favoriser la réflexivité	« Quelles questions pourrais-je poser à un dev démotivé ? »	Question socratique, Reformulation contextuelle
Documenter une intervention	« Rédige un retour synthétique de cette séance de coaching. »	Résumé ciblé, Curateur de pratique

Trucs et tactiques

-  **Travailler à voix haute avec le LLM** : se laisser surprendre par ses propres formulations.
-  **Utiliser les motifs comme canevas** pour préparer une intervention (Exploration guidée, Clarification, Miroir...).
-  **Créer une base de prompts pédagogiques ou d'atelier**, utilisable en animation collective.
-  **Coacher un rôle avec le LLM** : simuler une posture, identifier ses angles morts.
-  **Prototyper une situation délicate avant d'y aller**, via une simulation dialoguée avec le LLM.

Points de vigilance

- Le LLM **ne perçoit ni l'émotion, ni le non-verbal** : c'est un outil, pas un être relationnel.
- Il **peut normaliser ou rationaliser** ce qui est de l'ordre du vivant, de l'émergent, de l'ambigu.
- Il ne remplace jamais **la présence, l'écoute active, le discernement humain**.

Pour aller plus loin

- Utiliser le LLM **comme partenaire d'intervision** : rejouer une situation pour la relire avec un autre angle.
- Introduire le LLM **en binôme avec un humain** dans un atelier, comme “participant augmenté”.
- Intégrer les motifs dans les **formations à l'agilité ou au coaching**, comme outils de posture augmentée.

Le coach augmenté n'est pas un automate de facilitation. C'est un humain qui choisit quand, comment et pourquoi entrer en conversation avec une IA. Il observe ses propres biais, déplace ses angles morts, et expérimente de nouveaux chemins... toujours au service de la relation et de la clarté collective.

Annexe 6 — Manager 3.0 augmenté : soutenir les dynamiques collectives avec un LLM

Le management 3.0 repose sur la confiance, la transparence, la délégation, la culture du feedback. Intégrer un LLM dans cette posture, c'est **ouvrir un espace de dialogue élargi**, pour mieux questionner, clarifier, décider... et parfois sortir de ses angles morts.

Objectif

Explorer comment un manager 3.0 peut **mobiliser un LLM comme partenaire de réflexion, de clarification, ou de préparation**, sans se déresponsabiliser — mais au contraire, en renforçant sa présence et sa lucidité.

Geste professionnel augmenté

Avant	Avec LLM
Préparer un entretien individuel	Simuler des scénarios de dialogue, tester des formulations
Clarifier une décision complexe	Explorer des critères, comparer des options
Animer une réunion	Générer des structures d'ordre du jour ou de facilitation
Prendre du recul sur une tension	Cartographier les causes possibles, reformuler les enjeux
Donner du feedback	Travailler la formulation, explorer plusieurs tons
Partager une vision ou une intention	S'appuyer sur le LLM pour synthétiser ou structurer ses idées

Postures activables

Posture	Description synthétique
Clarificateur stratégique	Reformule les intentions, structure la pensée
Éclaireur de tensions	Explore les causes et alternatives dans une situation sensible
Facilitateur de réunion	Prépare des séquences d'animation adaptées aux contextes
Designer d'intention	Met en forme une idée pour la rendre communicable et inspirante
Feedbackeur intentionnel	Choisit son ton, ses mots, le moment, avec nuance et justesse
Sparring partner réflexif	Utilise le LLM comme miroir pour mieux se positionner

Exemples de situations courantes

Situation	Prompt-type	Motifs activés
Préparer un feedback difficile	« Aide-moi à formuler un feedback sur ce comportement observé. »	Question socratique, Reformulation par impact
Clarifier un rôle flou	« Voici le contexte. Propose une version claire de ce rôle. »	Spécification inversée, Synthèse structurée
Gérer un conflit entre deux coéquipiers	« Quelles pistes d'explication possibles pour cette tension ? »	Arbre des causes, Miroir de perception
Préparer un 1:1 stratégique	« Quelles questions puissantes poser à ce collaborateur ? »	Exploration guidée, Curiosité socratique
Communiquer une vision	« Propose plusieurs formulations de cette intention managériale. »	Co-conception, Reformulation inspirante

Trucs et tactiques

-  **Faire parler le LLM à la place d'un rôle ou d'un point de vue** : "Si j'étais le collaborateur, comment pourrais-je recevoir ça ?"
-  **Demander une carte d'options ou de scénarios** face à une situation ambiguë.
-  **Travailler plusieurs formulations** d'un message pour en affiner le ton.
-  **Relire ses propres décisions à travers un miroir IA** : "Quelles hypothèses implicites ?"
-  **Tenir une base de prompts managériaux efficaces** (1:1, feedback, annonces, etc.)

Points de vigilance

- Le LLM n'a ni ressenti, ni connaissance du terrain : il éclaire, mais ne décide pas.
- Il peut produire des suggestions socialement polies mais culturellement inadaptées.
- Trop s'en remettre au modèle peut affaiblir la responsabilité managériale.

Pour aller plus loin

- Intégrer le LLM dans une préparation collective de décisions (comité, atelier...).
- Proposer au LLM de jouer un rôle en atelier, pour diversifier les perspectives.
- Utiliser le LLM comme outil d'auto-coaching entre deux points de management.

Le manager augmenté n'automatise pas ses gestes : il les affine, les éclaire, les met à l'épreuve. Il reste garant du cadre et du sens, mais accepte de penser à plusieurs voix — y compris avec une machine. C'est un manager qui écoute plus finement... pour décider plus justement.

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

L'avènement des modèles de langage (LLM) ne se limite pas à une révolution technique : il redéfinit en profondeur nos façons de penser, de créer, de collaborer. Développeur, architecte, enseignant, coach ou manager : chacun voit son rôle évoluer vers une nouvelle forme d'artisanat — à la croisée du langage naturel, de l'intelligence artificielle et du design itératif.

Ce livre propose un **langage de motifs conversationnels**, inspiré de l'approche de Christopher Alexander, pour structurer ces nouvelles pratiques. Chaque motif décrit une situation typique, une tension récurrente, une solution praticable et ses effets. Vous apprenez à :

- formuler des prompts clairs et robustes,
- co-concevoir avec un LLM dans des contextes variés,
- affiner des architectures, des tests ou des idées métier,
- renforcer vos postures de facilitateur, de curateur, d'explorateur...

Un ouvrage hybride, comme nos métiers

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 explorent déjà, ou veulent oser demain.

Écrit à deux voix : un humain, une IA

Né d'un dialogue constant entre **Samuel Bastiat**, praticien aguerri de l'agilité, du développement logiciel et de l'intelligence collective, et un modèle de langage de nouvelle génération, cet ouvrage est aussi un témoignage vivant de ce qu'il propose : une **écriture augmentée**, réflexive, itérative, parfois surprenante... toujours habité par le souci du sens.

Une invitation à changer de posture

Et si concevoir du logiciel devenait un art de la conversation ? Et si dialoguer avec une IA devenait un levier pour penser mieux, ensemble ? Et si nos métiers devenaient plus humains... à mesure que nos outils deviennent plus puissants ?

Bienvenue dans l'ère de la conception logicielle augmentée. Bienvenue dans un livre qui vous parle — au sens propre.