LLM-Assisted Software Design

Un langage de motifs pour les nouvelles pratiques de conception logicielle

Samuel Bastiat

Texte rédigé avec la collaboration d'un LLM (GPT-4.5)

Olivier Azeau

Idée originale

Édition numérique — 2025

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.

📚 Sommaire

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.

Chapitre 1 — Introduction : concevoir avec l'IA, un nouvel artisanat logiciel

- L'émergence des LLM dans le développement
- Pourquoi un *pattern language*
- Posture du développeur augmenté
- Anecdotes et mises en situation
- Un exemple introductif de motif : « Design dialogué »

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

- Du besoin flou au prompt structuré
- Techniques de reformulation
- Variantes de formats (bullet points, contraintes, scénarios)
- Exemples concrets de formulation selon les rôles (dev, PO, architecte)
- Illustration schématique

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

- Pourquoi parler de motifs
- Structure type d'un motif
- **Motif 1 : Question socratique**
- **Motif 2 : Dialogue exploratoire**
- **Motif 3 : Boucle de vérification**
- **Motif 4 : Refactorisation assistée**
- **Motif 5 : Prototypage supervisé**
- Carte mentale des motifs
- Illustration synthétique

Chapitre 4 — Rôles augmentés : les nouvelles postures dans une équipe avec LLM

- Le développeur concepteur de prompt
- Le tech lead comme modérateur sémantique
- Le PO augmenté : test d'intentions
- Scénarios de collaboration homme-IA
- Carte des rôles et responsabilités

Chapitre 5 — Architectures conversationnelles

- Co-concevoir une architecture
- Chaîner des prompts, versionner des intentions
- Créer des abstractions durables avec LLM
- Exemples d'architectures générées, validées et ajustées en dialogue
- Schéma de chaînes de prompts

Chapitre 6 — Qualité et éthique dans la boucle

- Promesses vs hallucinations : les risques connus
- Grilles d’évaluation humaine du résultat
- Mettre en place des revues de prompts
- Le développeur comme garant du sens produit
- Cas pratiques : audits, conformité, robustesse

Chapitre 7 — Intégrer l’IA dans les cycles agiles

- L’IA dans les cérémonies Scrum (planning, revue, rétro)
- Story mapping et prompting
- Estimation assistée, mais pas automatique
- Mesurer la valeur produite avec l’IA
- Exemples d’usages en sprint

Chapitre 8 — Apprentissages, documentation et transmission

- L’IA comme tuteur et partenaire pédagogique
- Génération assistée de documentation
- Prompts comme artefacts à versionner
- Ateliers d’équipe, learning loops et coaching augmenté

Chapitre 9 — Contextualisation et adaptation

- En startup, en DSI, en open source
- Cas d’usage sectoriels (santé, industrie, éducation)
- Contraintes réglementaires, sécurité, données
- Adapter les motifs au contexte

Chapitre 10 — Limites et points de vigilance

- Quand ne pas utiliser un LLM
- Les angles morts de l’automatisation
- Les biais, la dépendance technologique, la perte de savoir-faire
- Bonnes pratiques pour préserver l’autonomie humaine

Chapitre 11 — Conclusion : vers un manifeste du développement augmenté

- Récapitulatif des transformations
- Proposition de manifeste (v0.1)
- Pratiques à prolonger et contributions à venir
- Une vision collective de l’avenir du logiciel

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

Ce chapitre d'introduction pose les bases du livre et de son ambition : explorer comment les développeurs peuvent collaborer efficacement avec les modèles de langage. Les chapitres suivants présenteront un ensemble structuré de motifs, chacun illustrant une pratique clé de ce nouveau paradigme : comment formuler des prompts clairs et efficaces, comment intégrer les LLM dans des cycles de développement agiles, comment garantir la qualité du code généré, ou encore comment penser l'architecture logicielle à l'ère de l'intelligence artificielle. Chaque motif sera accompagné d'exemples concrets, d'astuces pratiques, d'erreurs fréquentes à éviter, et de variations selon les contextes techniques ou organisationnels. En fournissant un tel langage de motifs, ce livre veut répondre à une double exigence : donner des repères concrets à l'action, tout en cultivant une posture réflexive sur l'usage des outils d'IA.

> **Pourquoi un "pattern language" ?**
>
> L'idée d'un langage de motifs s'inspire du travail de l'architecte Christopher Alexander, qui proposait une grammaire de motifs pour la construction de lieux de vie harmonieux. Appliqué au développement logiciel, et plus encore à la co-conception assistée par LLM, ce modèle permet d'ancrer les bonnes pratiques dans des situations concrètes, tout en donnant la possibilité de les combiner, de les adapter et de les enrichir dans le temps. C'est une invitation à penser la technique comme un artisanat vivant, structuré mais jamais figé.

Le monde du développement logiciel est en mutation rapide. Les modèles de langage de grande taille (LLM, pour "Large Language Models") bouleversent profondément la manière dont les logiciels sont pensés, conçus, écrits et maintenus. Leur capacité à comprendre et générer du langage naturel ainsi que du code transforme radicalement le rôle du développeur, qui devient progressivement un chef d'orchestre de l'intention, du sens et de la qualité logicielle. Ce changement implique une redéfinition de la posture professionnelle, mais aussi de la manière dont les équipes collaborent et innovent.

> **Anecdote : Le tri selon l'IA**
>
> Prenons un exemple simple : autrefois, une tâche comme créer un algorithme de tri nécessitait de l'écrire de A à Z, en s'assurant de sa performance et de sa robustesse. Aujourd'hui, un LLM peut non seulement générer ce code, mais aussi proposer plusieurs variantes selon les contraintes de votre projet (langage, environnement, volume de données), le commenter, le tester, et même suggérer des optimisations ou alternatives plus adaptées. Lors d'un atelier avec des étudiants, une simple demande de

> tri personnalisé a ouvert la voie à une discussion sur la complexité algorithmique, les cas pathologiques, et les formats de visualisation — tout cela amorcé par un échange avec un LLM.

Ce livre propose un langage de motifs (*pattern language*) original, par exemple le motif "Design Dialogué" qui décrit comment dialoguer avec un LLM pour construire progressivement une architecture logicielle. Ce motif, que nous détaillerons plus loin, met en lumière la puissance de la co-construction guidée par questions-réponses itératives, permettant de révéler des angles morts dans le raisonnement ou d'explorer plusieurs options en parallèle. Il est conçu pour aider les équipes techniques à structurer leur collaboration avec ces nouveaux outils puissants. Il s'agit d'offrir un cadre, une culture, une boîte à outils adaptée à cette nouvelle époque du développement logiciel. Ces motifs sont des repères pour structurer les usages, comprendre les nouvelles pratiques, et éviter les pièges fréquents. Ils peuvent être utilisés aussi bien dans des contextes individuels (développeur en solo) que collectifs (équipe agile, pair programming, revues de code augmentées).

Il ne s'agit plus seulement d'utiliser l'IA comme un assistant ou une aide à l'autocomplétion, mais bien de co-construire avec elle des systèmes compréhensibles, testables, évolutifs et auditables. Le développeur moderne devient à la fois concepteur de prompts, arbitre sémantique et garant du sens produit. Par exemple, lors d'un projet récent de refonte d'un système de gestion documentaire, une équipe a utilisé un LLM pour explorer différentes options d'architecture. Le développeur jouait alors un rôle d'animateur du dialogue avec le modèle, reformulant les besoins, validant les hypothèses, et veillant à la cohérence technique globale. Ce va-et-vient entre humains et IA a permis non seulement de gagner du temps, mais aussi de révéler des choix que l'équipe n'aurait peut-être pas envisagés seule. Cette posture hybride, entre analyste, facilitateur et ingénieur, deviendra sans doute la norme dans les prochaines années.

Ce livre s'adresse aux développeurs, architectes, tech leads, coachs, enseignants, chercheurs et penseurs systémiques curieux d'explorer un avenir où le prompt devient aussi important que la syntaxe, et où le langage naturel est une interface de création logicielle à part entière. Il s'agit d'un guide, mais aussi d'un manifeste : pour une pratique du développement où l'humain et l'IA avancent ensemble, avec clarté, exigence et responsabilité. Nous croyons que cette synergie est porteuse d'un nouvel artisanat numérique, fait d'intelligence collective, de rigueur, et d'invention partagée.

La grammaire de l’intention : penser et formuler avec un LLM

Avant même de parler de prompts ou d’outils, il nous faut revenir à la nature même de la collaboration entre humain et LLM. À quoi ressemble une bonne interaction ? Comment reconnaître une dérive, une ambiguïté, une impasse ? Comment transformer une simple commande en un véritable dialogue productif ?

Un LLM ne pense pas. Il infère, il complète, il modélise des suites probables de mots. Ce fonctionnement statistique peut générer des perles… ou des absurdités. Le rôle de l’humain devient alors celui d’un médiateur du sens : il guide, corrige, affine, reformule. Ce rôle est loin d’être passif. Il implique une écoute attentive, une capacité à reformuler des attentes, une exigence dans la précision des formulations et une vigilance continue face aux biais ou aux incohérences possibles.

Imaginons une séance de conception entre deux développeurs, sauf que l’un d’eux est un LLM. Ce partenaire connaît tout le Web, mais ignore votre contexte exact. Il est rapide, mais naïf. Il est créatif, mais oublie parfois ce qu’il vient de dire. La grammaire de cette interaction, c’est l’ensemble des règles implicites et explicites que vous mettez en place pour que la collaboration reste productive. Ces règles ne sont pas figées : elles s’adaptent selon les outils, les objectifs et les habitudes des personnes impliquées.

Voici quelques principes émergents :

- Toujours recontextualiser.
- Poser une question à la fois.
- Demander des justifications.
- Réinjecter les décisions importantes dans la suite du dialogue.

Exemple : dans un projet d’application de gestion de stock, une équipe a utilisé un LLM pour explorer les choix entre architecture en microservices ou en monolithe modulaire. Le prompt initial était vague. Le LLM a généré un comparatif généraliste. En affinant, en injectant des contraintes spécifiques (taille de l’équipe, fréquence des déploiements, besoin d’évolutivité horizontale), les réponses sont devenues beaucoup plus pertinentes. C’est cette capacité à dialoguer, à cadrer, à itérer, qui fait la qualité de la collaboration. Le LLM devient alors un partenaire de simulation, un interlocuteur technique, voire un copilote décisionnel.

Ce chapitre introduit ainsi les grands gestes de cette grammaire nouvelle : cadrer, questionner, reformuler, résumer, contrôler. Cela suppose également une posture d’écoute active et de vigilance critique : il ne suffit pas d’obtenir une réponse, encore faut-il en interroger la pertinence, la source, la cohérence avec le reste du projet. Une équipe travaillant sur une application bancaire a ainsi développé une habitude utile : chaque fois qu’un LLM proposait une solution, l’un des développeurs reformulait cette solution sous forme de diagramme ou de cas d’usage, puis la soumettait au modèle pour validation ou extension.

Cette méthode, fondée sur la reformulation visuelle et le questionnement continu, s'est révélée très efficace pour détecter des erreurs subtiles ou des raccourcis dangereux. Elle invite à articuler langage technique et visualisation pour renforcer la robustesse des idées générées.

Un autre exemple intéressant vient d'un atelier de conception participative, où un LLM était utilisé comme facilitateur d'idéation. Les participants posaient chacun leur tour une question au modèle, puis débattaient des propositions émises. Loin de figer la discussion, l'IA servait ici de catalyseur, rendant visibles des pistes implicites ou relançant la créativité collective. Certains motifs proposés par le LLM ont même été repris et raffinés collectivement, montrant une hybridation fertile entre intelligence humaine et intelligence artificielle. Cela montre bien que la grammaire de l'interaction n'est pas une technique froide, mais un art de l'ajustement : ajustement du langage, des intentions, des formats de réponse attendus, et surtout ajustement mutuel entre ce que l'on cherche et ce que l'outil peut offrir. Chaque motif à venir s'inscrit dans ce mouvement, avec une attention particulière à la manière dont le langage façonne l'outil, et inversement. Comprendre cette grammaire, c'est poser les fondations d'un partenariat fructueux avec les LLM.

Les motifs du dialogue : construire un langage de conception avec les LLM

Dans ce chapitre, nous allons explorer les motifs fondamentaux qui émergent de l'interaction régulière entre un développeur et un LLM. Ces motifs sont les unités de base d'un nouveau langage de conception, façonné non plus uniquement par les frameworks ou les langages de programmation, mais par les récurrences d'usage entre l'humain et le modèle. Ils forment une bibliothèque de pratiques réutilisables, adaptables, que les équipes peuvent combiner, faire évoluer, et transmettre.

L'objectif est ici de mettre en lumière des structures de pensée récurrentes, des scénarios typiques, et des séquences d'interaction efficaces. Par exemple :

- Comment amorcer un dialogue avec un LLM quand on part d'une idée floue ?
- Comment affiner un besoin technique à travers plusieurs itérations ?
- Comment structurer la co-écriture de code, de tests ou de documentation ?
- Comment obtenir une aide sur une impasse ou une panne conceptuelle ?

Ces situations sont autant d'occasions de formaliser des motifs. Nous les présenterons à travers une fiche type comprenant : nom, contexte, problème, solution, conséquences, et exemples. Le tout vise à construire un langage partagé qui outille les praticiens, tout en renforçant leur posture critique.

Motif 1 : « Question Socratique » — Reformuler pour comprendre

Contexte : Vous êtes confronté à un besoin exprimé de manière floue ou partielle (ex. : "Je veux faire une API pour envoyer des alertes en cas d'erreur système"), ou vous devez explorer un domaine que vous maîtrisez peu.

Problème : Un prompt trop vague entraîne souvent une réponse générique, peu exploitable ou déconnectée du contexte réel. Or, les LLM brillent lorsqu'ils sont alimentés en contraintes, en contexte, et en objectifs précis.

Solution : Engagez un dialogue progressif en reformulant la demande initiale sous forme de questions. Par exemple :

- "Quels types d'erreurs souhaitez-vous capturer ?"
- "Les alertes doivent-elles être transmises en temps réel ou agrégées ?"
- "Quel canal de notification privilégier (email, Slack, webhook) ?"

Cette technique, inspirée de la maïeutique socratique, pousse le demandeur (vous-même ou un collègue) à clarifier ses intentions, ce qui améliore la qualité du prompt et donc la pertinence de la réponse.

Conséquences :

- Le besoin devient plus clair, même pour les humains impliqués.
- Le prompt s'enrichit naturellement à chaque itération.
- Le LLM agit comme un catalyseur de réflexion, pas simplement comme un moteur de complétion.

Exemple : Dans un projet d'alerte météo automatisée, une équipe a d'abord demandé au LLM de "créer un script d'envoi d'alertes". La réponse obtenue ne convenait pas : elle reposait sur des hypothèses erronées (envoi par email, langage Python, etc.). En reformulant progressivement les besoins (fréquence, canal, gestion des seuils de gravité, intégration dans l'architecture existante), le LLM a fini par générer une solution beaucoup plus proche du besoin réel. Cette posture a ensuite été institutionnalisée dans l'équipe sous la forme d'un atelier régulier de co-design assisté par LLM.

Motif 2 : « Exploration Guidée » — Décomposer pour mieux avancer

Contexte : Vous devez concevoir une fonctionnalité complexe, découvrir un domaine technique inconnu, ou explorer une solution potentielle dont les tenants et aboutissants vous échappent encore.

Problème : Lorsque vous posez une question large ou trop abstraite à un LLM (ex. : "Comment concevoir une API REST sécurisée ?"), la réponse obtenue peut être trop généraliste, peu exploitable, voire source de confusion. Le LLM donne l'impression de « savoir tout faire » mais sans guider efficacement le cheminement.

Solution : Adoptez une posture d'exploration progressive en demandant au LLM de découper la tâche ou le domaine en étapes claires. Utilisez des formulations comme :

- "Peux-tu me décrire les grandes étapes pour… ?"
- "Quelle serait une première version minimale (MVP) ?"
- "Quels choix technologiques sont à envisager, et pourquoi ?"
- "Propose un plan d'implémentation par étapes."

Le but est de transformer une montagne confuse en une série de collines franchissables.

Conséquences :

- Le LLM joue un rôle de guide structurant, aidant à prioriser les étapes.
- La charge cognitive est réduite, car vous avancez pas à pas.
- Le dialogue devient un levier de formation continue : chaque étape est une occasion d'apprentissage ciblé.

Exemple : Dans un projet de migration d'une application monolithique vers une architecture en microservices, une développeuse ne savait par où commencer. Plutôt que de demander "Comment faire la migration ?", elle a utilisé l'exploration guidée : "Peux-tu me proposer un plan en cinq grandes étapes pour migrer un monolithe vers des microservices, avec les pièges courants à éviter ?" Le LLM a alors proposé une approche itérative, incluant la cartographie des dépendances, la séparation des contextes métier, la mise en place d'une couche d'API Gateway, et des stratégies de test spécifiques. Chaque étape a ensuite été

discutée plus en détail, facilitant la prise de décision et la coordination avec l'équipe.

Une session de pairing augmentée

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

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

Le LLM répond :

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

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

Motif 3 : « Spécification Inversée » — Obtenir un plan à partir du code

Contexte : Vous découvrez un morceau de code existant, hérité ou généré automatiquement, mais dont les intentions ne sont pas documentées ou pas comprises. Vous devez le maintenir, le refactorer ou le réécrire, mais vous manquez de vision sur sa finalité, ses contraintes ou sa logique métier.

Problème : Lorsque le code précède la conception (cas fréquent avec les LLM), il devient difficile d'en déduire les intentions originales. Cela ralentit la compréhension, multiplie les erreurs et fragilise la maintenabilité. Le risque est de bricoler sans fond, ou de produire de la dette technique involontairement.

Solution : Utilisez le LLM pour reconstituer les intentions implicites à partir du code : faites-lui "remonter" les spécifications à partir d'un extrait de code donné. C'est l'inverse du schéma habituel (besoin → code) : ici, on produit la documentation ou le plan de test *à partir* du code. Vous pouvez formuler des demandes comme :

- "Peux-tu me décrire ce que ce code est censé faire, étape par étape ?"
- "Quelles règles métiers peux-tu déduire de ce traitement ?"
- "Peux-tu générer une documentation technique à partir de ce fichier ?"
- "Quelles hypothèses ce code semble-t-il faire sur les données ?"

Conséquences :

- Le code devient un point d'entrée vers la compréhension métier.
- Les équipes peuvent mieux documenter des systèmes anciens ou mal maintenus.
- Cela permet de générer des cas de test à partir de l'implémentation existante.

Exemple : Une développeuse reprend un ancien script Python d'analyse de logs réseau, sans documentation ni tests. Elle copie le code dans un prompt et demande au LLM : *"Explique-moi ligne par ligne ce que fait ce script et quelles règles d'analyse il applique."* Le LLM identifie plusieurs étapes (filtrage par IP, regroupement par heure, détection d'anomalies par seuil), qu'il reformule sous forme d'un pseudo-algorithme. La développeuse peut alors générer une série de tests unitaires, identifier une condition oubliée (filtres sur protocoles), et décider de refactorer le tout sous forme de classes. Le modèle a joué ici le rôle de traducteur et de cartographe inversé.

> **Un LLM comme auditeur de code**
>
> Lors d'un audit technique sur un système de facturation, une équipe se retrouve face à un module PHP de plus de 800 lignes, écrit il y a 8 ans, sans tests ni documentation. Plutôt que de l'analyser ligne par ligne, l'équipe décide de le soumettre au LLM par blocs successifs.
>
> À chaque itération, le prompt est : *"Quels traitements réalise ce bloc de code ? Quelle règle métier cela semble-t-il implémenter ?"*
>
> Le LLM identifie la détection de doublons, la vérification de TVA, la gestion d'arrondis, et les cas particuliers de facturation croisée. Ces éléments sont mis en parallèle avec les processus métier décrits dans la documentation client, révélant plusieurs écarts non documentés.
>
> Cette approche de spécification inversée a permis de :
>
> - Documenter rétroactivement un système critique,
> - Réconcilier la logique métier et l'implémentation réelle,
> - Planifier une refonte progressive sans repartir de zéro.

🧿 Variantes du motif « Spécification Inversée »

📌 Variante 3.1 : *Reconstruction d'User Stories*

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

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

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

📌 Variante 3.2 : *Déduction des Hypothèses*

Demandez au LLM :

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

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

📌 Variante 3.3 : *Contrat implicite*

Demandez :

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

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

🛠 Mises en pratique concrètes

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

Prompt-type :

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

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

✓ Pratique 2 : Écrire la documentation avant refactorisation

Demande :

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

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

✓ **Pratique 3 : Détecter les effets de bord**

Prompt :

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

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

🔄 Intégration dans un workflow d'équipe

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

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

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

Posture recommandée

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

Motif 4 : « Modèle Miroir » — Comparer plusieurs versions pour éclairer un choix

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

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

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

Prompt-type :

- *« Propose-moi 3 variantes de cette fonction avec des styles ou des structures différentes. Puis compare-les sur lisibilité, performance, testabilité. »*
- *« Voici deux options d'architecture microservices. Peux-tu me les comparer sur les plans de la résilience, de la scalabilité, et de la complexité opérationnelle ? »*

Conséquences :

- On décentre le raisonnement : ce n'est plus "la bonne solution", mais "la plus adaptée au contexte".
- Cela entraîne une meilleure explicitation des critères de choix.
- Le LLM devient partenaire d'un raisonnement dialectique, pas seulement un fournisseur de contenu.

Exemple concret

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

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

Le LLM est sollicité via ce prompt :

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

Le modèle identifie que :

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

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

> **Encadré : Le LLM comme miroir d'équipe**
>
> Dans une réunion de design technique, une équipe ne parvient pas à se mettre d'accord entre deux styles de validation de formulaire côté frontend : impératif (en JS pur) ou déclaratif (via une lib type Formik ou React Hook Form).
>
> Un développeur propose de demander au LLM une comparaison structurée.

> Le prompt devient : *« Compare les styles impératif et déclaratif de validation de formulaire côté React. Donne des exemples de code et compare sur maintenabilité, UX et facilité de test. »*
>
> Le LLM répond avec clarté, exemples à l'appui, révélant que le choix dépend du niveau de complexité métier attendu et de l'organisation du code. Cela permet à l'équipe de ne pas trancher "par opinion", mais sur des critères explicites.
>
> Le modèle a servi ici de **facilitateur de clarification technique collective**, sans remplacer la décision humaine.

🔁 Variantes du motif « Modèle Miroir »

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

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

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

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

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

- "Quel cas d'usage pourrait faire échouer cette architecture ?"
- "Donne-moi un exemple où ce code produit un comportement inattendu."
- "Y a-t-il un scénario qui rendrait ce modèle inefficace ?"

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

Conséquences :

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

la documentation des limites d’un système.

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

> 📌 **Encadré — Posture réflexive : oser douter du bon élève**
>
> L’une des illusions les plus tenaces dans l’usage des LLM est celle de la "réponse parfaite" dès la première itération. Un motif comme "Clarification par contre-exemple" invite à adopter une posture scientifique : tester, falsifier, chercher ce qui ne va pas, même quand tout semble aller bien. Cela s’apparente à une relecture interne du raisonnement — une forme de revue de code dialoguée — où le développeur devient enquêteur des failles possibles. C’est aussi une manière de former les plus jeunes à ne pas confondre autorité de l’outil et vérité absolue.

Anatomie d’un bon prompt : précision, contexte et intention

Les performances des modèles de langage ne dépendent pas uniquement de leur puissance technique, mais surtout de **la qualité de l’interaction** qu’on construit avec eux. Et au cœur de cette interaction se trouve l’art du *prompt*. Ce chapitre propose une plongée dans la construction de bons prompts, en analysant leurs composants essentiels, en montrant des exemples concrets, et en définissant des motifs récurrents d’écriture.

Pourquoi ce chapitre ?

Trop souvent, on pense qu’un prompt est une simple question. Mais un bon prompt est en réalité un **acte de design**, une manière de structurer la pensée, de poser le cadre, de transmettre une intention. Il s’apparente à une interface entre deux intelligences : humaine et artificielle.

Trois dimensions fondamentales

1. **Précision** : éviter les formulations vagues, ambigües ou multi-interprétables. → Ex : "Donne-moi un code Python" → **trop large** ✓ Préférer : "Écris une fonction Python qui trie une liste de dictionnaires par une clé 'date', en ordre décroissant."

2. **Contexte** : fournir les éléments utiles pour cadrer la réponse : langage, environnement, style, contraintes métier… → Ex : "Je développe une API REST en Node.js dans un contexte de microservices gérés par Docker."

3. **Intention** : exprimer clairement le *but* visé, pas seulement la tâche. → Ex : "Je veux un script Shell pour automatiser le déploiement, afin que même un stagiaire puisse l’exécuter sans rien casser."

> 📌 **Encadré — Le prompt n’est pas une requête, c’est une conversation dirigée**
>
> Il est utile de penser le prompt comme une amorce de conversation, pas comme un ordre. Le prompt bien conçu contient souvent une *dynamique* : il prépare la suite du dialogue. Un bon prompt anticipe les rebonds, les vérifications, les approfondissements. Il ouvre l’espace d’échange au lieu de le fermer.

Typologie des prompts efficaces

Nous proposons ici une typologie structurée, illustrée de motifs que l’on retrouvera tout au long du livre :

- **Prompt "Contexte + Tâche"** :

> "Dans le cadre d'un service d'authentification OAuth2 en Go, écris un middleware qui vérifie la présence d'un token JWT valide."

- **Prompt "Exemple + Variation"** :

> "Voici une fonction JavaScript pour filtrer un tableau. Peux-tu proposer une version plus performante avec `reduce` ?"

- **Prompt "Roleplay"** :

> "Agis comme un expert Django senior. Donne-moi les étapes clés pour refactorer une app monolithique en microservices."

- **Prompt "Pas-à-pas"** :

> "Explique-moi étape par étape comment sécuriser une API avec des jetons CSRF, comme à un étudiant de niveau bac+2."

Bonnes pratiques et erreurs fréquentes

✓ **Bonnes pratiques** :

- Être explicite sur les contraintes : langage, version, bibliothèque cible.
- Utiliser le formatage (listes, bullet points, code blocks) pour structurer la demande.
- Préciser le niveau de détail attendu : résumé, tutoriel, snippet, code complet, benchmark ?

✕ **Erreurs fréquentes** :

- Poser plusieurs questions en une.
- Employer des termes vagues : "optimiser", "simplifier", "améliorer" — sans dire ce qu'on entend par là.
- Oublier le *pourquoi* de la demande.

Exemple comparatif

Prompt faible :

> "Fais-moi une API Node."

Résultat : réponse générique, non contextualisée.

Prompt amélioré :

> "Je veux créer une API REST en Node.js avec Express. Elle doit permettre de créer, lire, mettre à jour et supprimer des utilisateurs stockés dans une base MongoDB. Je veux du code modulaire, avec une bonne séparation des responsabilités, sans ORM. Peux-tu me proposer la structure de fichiers et le code de base pour démarrer proprement ?"

Résultat : réponse structurée, adaptée, directement exploitable.

Fiche-outil — Anatomie d'un bon prompt

🎯 Objectif

Concevoir un prompt efficace pour interagir avec un LLM dans un contexte de développement logiciel, en maximisant la pertinence et l'utilité des réponses.

📐 Structure type d'un prompt efficace

| Élément | Description | Exemple | | | -- | | | **Contexte** | Donne le cadre technique, fonctionnel ou organisationnel | "Je travaille sur une API REST en Python avec FastAPI, déployée sur AWS Lambda…" | | **Tâche claire** | Décrit précisément ce que vous attendez | "…je veux une fonction pour vérifier un JWT dans les headers d'une requête HTTP." | | **Contraintes** | Précise les choix technos, limites ou préférences | "Sans utiliser d'ORM, et avec des logs clairs en cas d'échec de validation." | | **Intention** | Fait apparaître le *pourquoi* de la demande | "Je veux que ce soit simple à comprendre pour un développeur junior." | | **Format attendu** | Indique le type de réponse souhaitée | "Peux-tu me donner un exemple commenté + les tests unitaires correspondants ?" |

📎 Astuces pratiques

- **Soyez spécifique** : un prompt générique donne une réponse générique.
- **Pensez séquence** : un bon prompt n'est que le premier pas d'un échange.
- **Nommez vos contraintes** : langage, bibliothèque, niveau de détail.
- **Ajoutez des exemples** : un exemple concret inspire une meilleure réponse.

⚠ À éviter

- ✕ Phrases trop générales : "fais-moi un code", "aide-moi avec ce bug"
- ✕ Absence de contexte : pas de langage, pas d'architecture, pas de but
- ✕ Prompts fourre-tout : trop d'idées mélangées, pas de hiérarchisation

Exemple comparé

Prompt faible	Prompt amélioré

"Donne-moi un code pour une API Node.js"	"Crée une API REST en Node.js avec Express, qui gère des utilisateurs stockés dans MongoDB. Structure le code en suivant une architecture MVC, sans ORM. J’ai besoin des routes CRUD, d’une validation d’entrée, et de quelques tests unitaires."

Cartographier les usages : typologie des interactions LLM-développeur

Dans ce chapitre, nous proposons une lecture transversale des motifs précédemment présentés en les reliant à des **situations-types** que rencontrent les développeurs dans leur quotidien. Il ne s'agit plus uniquement de parler en termes de « patterns » abstraits, mais de comprendre **quand et pourquoi** tel ou tel motif s'active, en fonction de l'intention du moment, du contexte de travail, ou encore du niveau de maturité de l'utilisateur avec les LLM.

Nous allons ainsi esquisser une **cartographie des usages** qui aide à naviguer dans les interactions avec les modèles de langage. Cette typologie offre aux équipes une meilleure lecture de leurs pratiques et permet aux formateurs, coaches et tech leads d'identifier les compétences associées à chaque posture.

1. L'explorateur — Interagir pour comprendre un domaine

> **Objectif :** recueillir des informations, structurer une compréhension initiale, identifier des angles d'approche.

L'explorateur formule des questions larges, cherche à définir un périmètre, à obtenir une vue d'ensemble d'un sujet. Il active souvent les motifs « Question Socratique » ou « Reformulation Itérative ». Ce type d'interaction est fréquent en début de projet, en phase de cadrage ou lors de l'arrivée sur un domaine inconnu (nouvelle techno, architecture existante à auditer, etc.).

Exemple : « Quels sont les principaux types de base de données NoSQL et dans quels cas les utiliser ? »

2. Le praticien opérationnel 🛠 — Résoudre un problème concret

> **Objectif :** générer ou corriger du code, automatiser une tâche, proposer une implémentation.

Cette posture est la plus répandue. L'utilisateur cherche une solution immédiate à un blocage technique, une aide à l'écriture, ou un gain de temps. Il utilise souvent les motifs « Co-écriture par reformulation », « Contre-exemple » ou « Prompt Contexte + Tâche ».

Exemple : « Génère une fonction Python qui détecte les doublons dans une liste de dictionnaires. »

3. Le concepteur structurant — Faire émerger une architecture ou un design

> **Objectif :** co-construire une vision technique cohérente à partir d'éléments épars.

Ici, le LLM est utilisé comme partenaire de réflexion. Le développeur ne cherche pas une solution, mais une structure, une articulation d'idées. Cela implique un travail en itérations, avec évaluation d'alternatives, scénarios, et documentation. Les motifs « Design Dialogué », « Exploration Parallèle » ou « Hiérarchie Intentionnelle » (chapitres suivants) y sont fréquents.

Exemple : « Quels modèles d'architecture sont adaptés à une application de messagerie sécurisée en temps réel ? »

4. Le pédagogue réflexif 🧒🏫 — Se former ou former à travers le dialogue

> **Objectif :** utiliser le LLM comme outil d'apprentissage, de transmission ou de formalisation.

Ce profil s'adresse souvent aux enseignants, mentors, ou aux développeurs en formation. Le LLM est utilisé pour expliquer, reformuler, illustrer, simuler des erreurs. Les motifs comme « Question Socratique », « Cas Limite », ou « Décomposition Progressive » sont ici clés.

Exemple : « Explique-moi pourquoi le mot-clé `await` est obligatoire dans une boucle for async en JavaScript. »

> 🗺️ **Carte synthétique à venir : correspondance motifs ↔ postures ↔ situations**

Cette cartographie permet d'identifier la pluralité des approches et de mieux comprendre que les LLM ne sont pas simplement des outils « génériques », mais des partenaires capables d'adapter leur réponse à la posture intellectuelle de l'utilisateur. Il devient alors possible de développer une véritable **intelligence d'usage**, c'est-à-dire une capacité à activer le bon levier au bon moment.

Intégrer les motifs dans le quotidien du développement

Les motifs que nous avons présentés jusqu'ici (dialogue socratique, clarification incrémentale, traduction de contexte, test guidé, etc.) sont puissants, mais leur impact dépend largement de la manière dont ils sont intégrés dans les pratiques existantes. Ce chapitre aborde la transition entre l'inspiration théorique et l'application quotidienne : comment faire de ces motifs une routine naturelle, fluide, collective et évolutive dans les environnements de travail réels ?

Du motif à la pratique : l'enjeu de l'appropriation

Il ne suffit pas de connaître les motifs ; encore faut-il les reconnaître dans les situations vécues, savoir quand et comment les activer. Cette appropriation passe par plusieurs leviers :

- **La ritualisation** : instaurer des moments spécifiques pour pratiquer (ex. : ateliers de co-prompting hebdomadaires, revues de prompts lors des revues de code).
- **La documentation vivante** : conserver les échanges les plus riches avec les LLM sous forme de fiches ou de journaux de conversation annotés.
- **Le retour d'expérience partagé** : échanger entre pairs sur les stratégies d'interaction, les ratés, les surprises. Le motif devient alors un objet de conversation, de transmission et d'apprentissage.

> **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.

Illustration : Le tableau de bord des motifs

Un tableau de bord interactif a été mis en place dans une entreprise pour suivre l'usage des motifs au fil des sprints. Il permet à chaque développeur de noter les motifs utilisés, leur efficacité perçue, et les suggestions d'amélioration. Ce retour quantitatif et qualitatif aide à identifier les motifs "oubliés", ceux qui méritent d'être enrichis, ou les moments clés où l'IA est mobilisée. Ce type de suivi, sans être contraignant, nourrit une intelligence collective sur les pratiques.

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

L'intégration des LLM dans le développement logiciel ne se limite pas à de nouveaux outils ou techniques : elle transforme en profondeur les rôles, les responsabilités et les dynamiques au sein des équipes. Le développeur devient à la fois facilitateur de sens, architecte de dialogue, évaluateur de qualité sémantique. Ce chapitre explore comment les métiers évoluent dans ce contexte d'augmentation cognitive, et quelles nouvelles compétences deviennent centrales.

Le développeur augmenté : un chef d'orchestre du raisonnement

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

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

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

De nouveaux rôles dans l'équipe

Plusieurs fonctions émergent ou évoluent fortement dans les équipes utilisant les LLM :

- **Prompt Designer** : expert dans l'art de structurer les requêtes selon les objectifs, les contraintes et le contexte.
- **Curateur de Connaissance** : personne chargée d'enrichir les modèles avec du contexte local (code, glossaire, politiques internes) pour augmenter leur pertinence.
- **Facilitateur de Dialogue** : rôle pivot dans les ateliers de co-conception assistée par IA, animant les itérations et veillant à l'équilibre humain-machine.
- **Auditeur Sémantique** : spécialisé dans la relecture critique des propositions IA, pour valider la cohérence, l'éthique et la sécurité.

> **Illustration simple** : Un tableau à double entrée avec les rôles traditionnels (Développeur, Architecte, Product Owner, Coach Agile…) et leurs évolutions en contexte augmenté (Prompteur, Animateur IA, Curateur sémantique…).

Des compétences transversales en émergence

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

- **La formulation explicite** (écrire ce qu'on pense, penser ce qu'on écrit).
- **L'interrogation stratégique** (poser les bonnes questions pour faire avancer le raisonnement de l'IA).
- **La navigation dans les incertitudes** (évaluer rapidement la fiabilité d'une réponse et pivoter si nécessaire).
- **La médiation entre humains et IA** (savoir retranscrire la complexité métier dans un langage compréhensible par un modèle).

Cas concret : une rétrospective augmentée

Dans une équipe de développement, le Scrum Master a proposé d'utiliser un LLM comme co-facilitateur de la rétrospective. Les membres de l'équipe formulaient les irritants ou les réussites du sprint, et le modèle proposait des regroupements, des axes de réflexion, voire des pistes d'action. Le rôle du Scrum Master a évolué : il ne centralisait plus les idées, mais orchestrai*t 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.

Responsabilité, transparence et limites : une éthique du développement augmenté

Le recours aux LLM dans le développement logiciel ne pose pas uniquement des questions de productivité ou d'efficacité. Il soulève aussi des enjeux éthiques profonds : qui est responsable du code généré ? Comment éviter les biais ? Que faire lorsqu'un modèle propose une solution trompeuse, non sécurisée ou inadaptée ? Ce chapitre aborde la dimension éthique de cette co-construction avec des intelligences statistiques, pour une pratique plus consciente et plus responsable.

La tentation de la délégation

La qualité apparente des réponses générées par les LLM peut créer une illusion de maîtrise ou d'autorité. Pourtant :

- Les modèles n'ont **ni compréhension réelle** ni conscience du contexte métier ou humain.
- Ils peuvent générer des réponses fausses avec une grande assurance linguistique.
- Ils sont **sensibles aux biais présents dans leurs données d'entraînement**, souvent invisibles à l'œil nu.

> **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.

Qui est responsable ?

Face à ce genre de situation, plusieurs questions deviennent centrales :

- Le développeur qui copie-colle un code généré sans le tester est-il responsable ?
- L'équipe doit-elle tracer les morceaux de code issus d'une IA ?
- Peut-on considérer un LLM comme une source de documentation ou comme un contributeur anonyme ?
- Que deviennent les exigences de conformité réglementaire (ex. : RGPD, cybersécurité, accessibilité) dans un monde de suggestions automatiques ?

Une approche responsable repose sur **l'explicitation des intentions, des choix et des arbitrages humains** à chaque étape. L'IA est un outil, pas un auteur.

Des pratiques pour une éthique active

Voici quelques pratiques émergentes pour renforcer l'éthique dans les usages :

- **Tracer l'origine des suggestions IA** dans les revues de code ou les commit messages.
- **Demander systématiquement des justifications** aux réponses générées : "Pourquoi cette solution plutôt qu'une autre ?"
- **Inclure des tests, validations ou relectures humaines** pour tout code ou contenu produit par IA.
- **Favoriser la diversité des points de vue** en confrontant les propositions IA à celles de l'équipe.
- **Documenter les prompts sensibles** ou ayant un impact critique (sécurité, données, logique métier complexe).

> **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.

Les limites des LLM : mieux les connaître pour mieux les utiliser

Il est essentiel de garder à l'esprit certaines limites structurelles des modèles de langage actuels :

- **Ils ne raisonnent pas** : ils extrapolent des séquences probables.
- **Ils n'ont pas de mémoire de long terme** sans l'ajout explicite de contexte.
- **Ils peuvent halluciner** (inventions crédibles mais fausses), notamment dans les domaines techniques pointus.
- **Ils sont sensibles à la formulation** : un même problème mal posé peut produire une réponse erronée ou biaisée.

Connaître ces limites n'est pas un obstacle, mais une condition pour construire une relation saine et maîtrisée avec l'IA.

Scénarios prospectifs : vers une ingénierie conversationnelle générative

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

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

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

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

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

> **Extrait de dialogue fictif :**
>
> **Humain :** Je veux une API REST pour gérer un agenda partagé entre utilisateurs.
>
> **Agent archi :** Je propose une architecture hexagonale, avec stockage PostgreSQL et middleware auth. D'accord ?
>
> **Humain :** Ajoute une logique de permissions fines. Front, tu suis ?
>
> **Agent front :** Oui, je vais générer une interface React avec Tailwind, accès conditionnel selon rôle.
>
> **Humain :** Testeur, fais un plan de tests critiques sur les accès concurrents.
>
> **Agent test :** Génération en cours…
>
> **Humain :** Sémanticien, documente le schéma de droits en langage clair.
>
> **Agent sémanticien :** Voilà une première version.

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

2. Architecture générative pilotée par dialogue

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

Le concepteur discute avec un modèle qui :

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

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

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

3. Le design conversationnel comme forme de développement

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

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

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

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

Vers une nouvelle ingénierie : augmentée, exploratoire, réflexive

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

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

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

Cadres de mise en œuvre : ateliers, méthodes et rituels pour une pratique augmentée

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

1. Atelier "Design de prompt en équipe"

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

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

Déroulé :

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

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

2. Rituel "Daily du dialogue"

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

Format léger, quotidien ou hebdo :

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

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

3. Atelier "Cartographie des motifs de dialogue"

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

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

Exemples de motifs :

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

> **Sortie possible :** Un "grimoire des dialogues utiles", propre à l'équipe

4. Le jeu des prompts absurdes

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

Règles :

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

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

5. Référentiel d'équipe "LLM Ready"

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

- Bonnes pratiques de prompting
- Modèles de prompts testés
- Risques connus (ex. : hallucinations en matière de sécurité, mauvais découpage métier…)
- Grille d'évaluation des réponses (pertinence, sécurité, robustesse, cohérence)
- Niveaux de criticité : quand valider avec un humain, quand automatiser ?

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

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

Une ingénierie augmentée est aussi une ingénierie sociale

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

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

Conclusion : vers un manifeste du développement augmenté

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

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

Ce que nous retenons

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

Une posture nouvelle

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

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

Et maintenant ?

Voici quelques propositions pour prolonger la dynamique :

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

Un manifeste du développement augmenté (version 0.1)

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

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

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

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

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

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

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

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

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