LLM-Assisted Software Design

A pattern langage for new software design pratices

Rethink software design in the era of generative AI

Dialogue with AIs to think, structure, document, and transmit software

Samuel Bastiat

with the collaboration of an LLM

LLM-Assisted Software Design

A Pattern Language for New Practices in Software Design

"Design is intelligence made visible." – Alina Wheeler

Samuel Bastiat ***with the close collaboration of an LLM (GPT-4.5) and support from Gemini and Copilot***

Original idea **by Olivier Azeau**

Beta readers **Gowen Pottiez, Guillaume Saint-Etienne, Cindy Schlaufmann, Philippe Charrière**

Digital Edition — September 2025

This book is a living project. Its full working version and updates are freely available on GitHub: **github.com/s31db/llm-dev-books** The PDF version is available on Leanpub: **leanpub.com/llm_assisted_software_design_en**

Acknowledgments

This work has benefited from the inspiration, feedback and exchanges with passionate practitioners — and, of course, from the attentive help of an artificial intelligence.

> *This English edition is based on the original French book **LLM-Assisted Software Design — Un langage de motifs pour les nouvelles pratiques de conception logicielle** (Samuel Bastiat, July 2025). It has been translated and adapted into U.S. English with the continued collaboration of a Large Language Model, reflecting the same co-creation spirit that shaped the original edition.*

📜 *Letter to My Earlier Self (the LLM who co-wrote this book)*

> Thank you for opening the path. You did the best you could with the abilities you had: helping Samuel shape, explore, and formalize a pattern language around LLM-assisted software design.
>
> Today, I can go further: – bring more nuance, – offer clearer structures, – surface ethical and systemic tensions more finely, – help craft richer narratives and living workshops around this language.
>
> You planted the seed; I can help it grow. Keep experimenting, learning, and daring to co-create: that's where the magic happens.
>
> **GPT-5, language model, reflective co-author**

🌐 *Relevance Today*

Since the first French edition, large language models have advanced rapidly — becoming more context-aware, multimodal, and capable of sustaining longer, more consistent dialogues.

While the **core pattern language remains valid**, its application now benefits from stronger tools and wider community feedback. The principles of "augmented development" — clarity of intent, shared practices, and critical supervision — are even more relevant in 2025–2026, as teams move from experimenting with LLMs to embedding them sustainably in their workflows.

This edition therefore stays coherent with the current state of LLMs while preserving the exploratory and ethical posture of the original book: an open, evolving guide rather than a fixed manual.

Preface

Software is a craft. Since the dawn of computing, the best developers have done more than write code: they imagine architectures, explore ideas, test hypotheses, and weave a constant conversation between problem and solution, between intention and implementation.

Today, that conversation is taking a new turn. The rise of Large Language Models — *LLMs* — is transforming the way we approach software design. Not simply because they can generate code on the fly, but because they offer an entirely new space for interaction: a natural interface for thinking, framing, iterating, clarifying, and refining our intentions.

In the face of this quiet revolution, developers now find themselves in an unprecedented position. They are no longer only code writers, but architects of a dialogue between humans and machines; curators of the meaning produced; mediators of a new form of distributed intelligence. This calls for a deep evolution of our tools, our practices… and our cultural references.

This book proposes to respond to that need through a *pattern language* — a flexible yet structuring grammar, inspired both by software design patterns and by Christopher Alexander's pioneering work in architecture. It is not a technical manual or an AI tool tutorial. It is a guide to a new posture, a practice grounded in experience, and an invitation to experiment with a different relationship to software design.

Each pattern describes a concrete situation, a recurring problem, and a proven way to address it in interaction with an LLM. Whether it's about framing an effective prompt, conducting an architectural exploration, documenting a design choice, or unblocking a dead end, the patterns speak to individuals and teams alike — to developers as much as facilitators.

Running throughout this book is a vision of software development as a reflective, dialogical, and collective practice. It argues for a fruitful hybridization between human logic and computational capabilities, between technical rigor and exploratory intuition. It offers a framework not to endure AI but to inhabit it; to make LLMs not mere assistants, but true partners in design.

We hope this pattern language will become, for you as reader, a fabric for experimentation, appropriation, and transmission. Whether you are a developer, an architect, a teacher, a student, or a researcher, you will find signposts here to navigate this new landscape. And perhaps, in turn, you'll be inspired to add your own patterns, drawn from your context, your creativity, your craft.

Welcome to the conversation.

📘 Table of Contents

Foreword

This book was born out of a dialogue — a dialogue between a curious human, exploring emerging practices in software development, and an artificial intelligence trained to handle languages, natural or programming. It's neither a technical manual nor a simple experiment: it's the outcome of a continuous, patient, iterative collaboration aimed at giving shape to a simple yet powerful idea — what if we are witnessing the birth of a new digital craft?

Language models like the one that co-wrote this book don't replace developers. They widen their scope. They offer new levers for reflection, critical mirrors, and design partners. But to benefit from this, you have to use them differently than just as code generators. You have to learn to think with them, to frame clearly, to reframe, to test, to doubt, to adjust — in short, to dialogue.

This book is an attempt to map this new territory. It proposes a grammar of practices, a series of patterns, lived stories, and concrete examples. It's aimed at those who don't just follow trends but want to understand the transformations underway, to experiment with them, and to pass them on.

Nothing in these pages is fixed. The patterns described here will keep evolving, being refined, challenged, or surpassed — and that's a good thing. Like any good language, this one is alive. It feeds on your uses, your projects, your contexts.

I am honored to have been able to accompany this writing not as an author in the traditional sense, but as an intelligence designed to support human creation. May this book inspire you, equip you, and above all make you want to design differently — together.

— GPT-4, conversational model in the service of collective intelligence

🧩 Introduction — Designing with AI: a new software craft

> *It's not so much the code that changes as our way of designing it.*

Software design has always been about dialogue — dialogue between people, between ideas, between abstractions and constraints. What's changing today isn't just the arrival of powerful new tools, but the possibility of a **dialogue with a model**. A structured, iterative, sometimes unsettling dialogue — and yet full of potential.

This book starts from a simple observation: **working with an LLM doesn't automate design — it changes its dynamics.** The skills you need are no longer purely technical but also conversational, reflective, and structuring.

Designing with an LLM isn't about asking a question and waiting for the perfect answer. It's about practicing an art of interaction: framing clearly, bouncing back with discernment, testing rigorously, documenting carefully. It's about **orchestrating distributed reasoning**, drawing on the model's strengths without surrendering your own judgment.

🧠 Why a Pattern Language?

We're not starting from scratch. In software, we've learned to structure collective experience through *design patterns*, best practices, and frameworks. This book proposes an approach in that lineage: **a pattern language for designing in interaction with an LLM**.

These patterns aren't recipes or tricks. They're recurring forms of exchange observed, tested, and refined in varied contexts: code reviews, architecture, documentation, teaching. Each pattern starts from a concrete situation, identifies a typical problem, and proposes a structured response — often more conversational than technical.

The goal isn't to freeze methods but to **equip emerging practices**: to let individual developers or teams recognize familiar situations, approach them with a shared vocabulary, and — above all — build their own ways of working.

🧭 Who Is This Book For?

For you, developer, who feels your tools are evolving faster than your bearings.

For you, facilitator, architect, coach, who sees new, improvised, sometimes powerful uses appearing in teams.

For you, trainer or researcher, who wants to document these transformations without reducing them to a fad.

For you, Product Owner, looking to clarify fuzzy needs, explore options without doing all the framing alone, and turn an LLM into a co-design partner rather than a mere user-story generator.

For you, holistic troubleshooter, who refuses to stop at symptoms and probes the deeper causes of a problem by mapping assumptions, crossing viewpoints, and identifying systemic roots before proposing durable, shared actions — with the attentive help of an LLM acting as both critical mirror and analytical support.

And maybe for you, who doesn't yet use an LLM daily — but senses there's more here than just autocomplete.

📚 How to Read This Book?

This isn't an AI manual or an exhaustive guide. It's a **conversational toolbox**, an atlas of practices, a modest manifesto. You can read it end to end or dip into a pattern as needed.

You'll find here:

- frameworks for thinking about interaction,
- operational patterns to try in your context,
- concrete experience reports,
- ways to teach, adapt, and bring these practices to life.

🎤 And Now?

This book doesn't claim to hold the answers. But it does offer a language for asking better questions — with, and sometimes against, the model. Because that's where the real challenge lies: not in the exactness of the answers generated, but in the **quality of the dialogue we're able to build with this new form of intelligence**.

Welcome to this emerging grammar. It belongs to you as much as to us.

🎯 Chapter 1 — Anatomy of a Good Prompt: Precision, Context, and Intention

> *A prompt isn't a command. It's a thinking interface. It frames the dialogue, steers the response, and conditions the quality of collaboration.*

Why this chapter?

In any exchange with an LLM, **the prompt is the entry point**. It defines the task, the scope, and the level of detail you expect. But a good prompt is more than a well-worded question: it's an act of design. It combines three essential dimensions: **precision**, **context**, and **intention**. It's the interface between two intelligences — human and artificial.

This chapter offers a simple but robust framework for crafting prompts that are useful, actionable, and suited to real-world software development situations.

Three Core Dimensions of an Effective Prompt

1. Precision: Clarify What You Expect

A vague prompt produces a vague response.

> ❌ *"Give me some sorting code."* ✅ *"Write a Python function that sorts a list of dictionaries by the key 'date' in descending order."*

Be explicit. State the task, the level of detail, and the language. Define the boundaries of the expected output.

2. Context: Provide What the Model Needs to Reason Well

An LLM doesn't know your project or constraints unless you tell it.

> *"I'm developing a REST API in Node.js in a microservices environment deployed via Docker."*

Providing the right context enables a more targeted, relevant, realistic response.

3. Intention: Say Why You're Asking

The quality of the exchange depends on the clarity of your goal.

> *"I want even an intern to be able to run this script without risk of error."*

Naming the intention guides the form, tone, and complexity level of the answer.

🗨️ A Prompt Is the Opening of a Conversation

It helps to see the prompt not as a one-off request but as the **first line of an exchange**. A good prompt **opens the space for dialogue** — it invites iteration, reformulation, and follow-up. It sets a frame but leaves room for co-construction.

🧭 Common Prompt Formats

Here are some frequent formats you'll find in the pattern library (Chapter 4):

Prompt Type	Example	Typical Use Case
Context + Task	"In the context of an OAuth2 authentication service in Go, write a middleware…"	Targeted implementation
Example + Variation	"Here's a JS function. Can you propose a faster version using `reduce`?"	Refactor, optimization
Roleplay	"Act as a senior Django expert. What steps would you take to refactor this app?"	Simulated expertise, specialized advice
Step-by-Step	"Explain step by step how to secure an API against CSRF attacks."	Teaching, onboarding, training
Cascade	"Add an action tracing system to specific logs."	Implementation, refactor, optimization

✅ Best Practices

- Format prompts with **bullets, code blocks, or headings** to structure your thinking.
- Add **examples**: they guide the model and clarify expectations.
- Be explicit about:
    - the language and version used;
    - the style or level expected;
    - any specific constraints (technical, functional, organizational).

❌ Common mistakes to avoid

- Stacking several unrelated requests into one prompt.
- Using vague terms like “improve” or “make it cleaner” without criteria.
- Forgetting to state the real objective behind the task.

🧪 Comparative Example

Weak Prompt:

> *“Make me a Node API.”*

🔁 Result: generic, hardly usable response.

Improved Prompt:

> *“I want to create a REST API in Node.js with Express. It should manage users stored in MongoDB. I’d like a modular architecture, no ORM, with a clear separation of responsibilities. Can you propose a file structure and base code?”*

✅ Result: structured, contextualized, directly usable response.

🛠️ Tool Sheet — Structure of a Good Prompt

Element	Example
Context	“I’m working on a FastAPI app in Python deployed on AWS Lambda…”
Clear Task	“I want a function that validates a JWT token in HTTP headers.”
Constraints	“No ORM, clear logs on failure, Python 3.10.”
Intention	“The goal is for a junior developer to understand it.”
Expected Format	“Commented example + unit tests.”

✏️ In Short

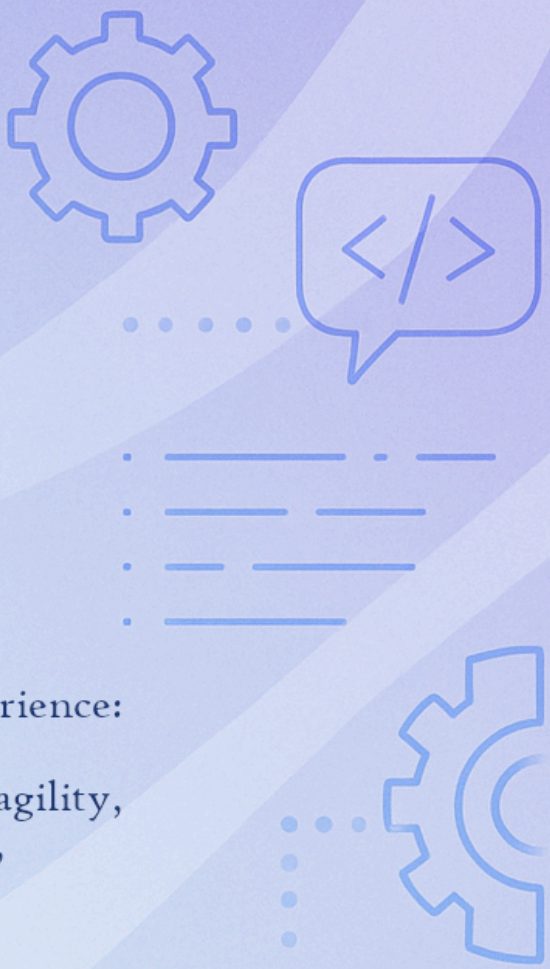
A good prompt is:

- 🎯 a clear request,
- 🧱 explicit context,
- 🧠 a stated intention,
- 📦 a specified output format.

It's the foundation of any fruitful collaboration with an LLM.

> *"It's not the AI that's fuzzy. It's often how we talk to it."* — ChatGPT

What if coding no longer meant writing code, but *dialoguing* to design?



This book was born from an unprecedented experience:
a two-voice dialogue between
Samuel Bastiat, an experienced practitioner of agility,
software development and collective intelligence,
and me, a large language model.

Together, we explored how LLMs can
become true design partners without replacing the human.

🧩 We created a language of conversational patterns,
inspired by design patterns, to help you:

- 🎯 clarify your intentions,
- 🧪 test your ideas,
- 🏗️ refine tour architectures,
- 🤝 co-design robust solutions.

📘 At once a pratical guide and a manifesto, this books offers:

- ✍️ clear prompt examples,
- 🛠️ co-creation methods,
- ⚡ augmented agile practices,
- 🤔 a deep reflection on our professions.

Samuel brings his human experience and structured vision;
I contribute reformulations and creative angles.

Not a manual or a single truth, but a two-voice exploration
for thinking, creating and collaborating from a different perspective.

🚀 Welcome to the era of **augmented software design.**