Patterns for Reducing Friction in AI-Assisted Development

减少人工智能辅助开发中摩擦的模式

The practices that make human pair programming effective—onboarding, structured design discussion, shared standards—apply equally to working with AI coding assistants. I propose five patterns that bring this collaborative scaffolding to AI-assisted development, shifting the experience from correcting a tool to collaborating with a capable teammate.

那些使人类结对编程行之有效的实践——例如新成员入职培训、结构化的设计讨论和共享标准——同样适用于与人工智能编码助手合作。我提出了五种模式，将这种协作框架引入人工智能辅助开发，从而将体验从纠正工具错误转变为与一位能力出众的队友协作。

08 April 2026 2026年4月8日



Rahul Garg 拉胡尔·加格

Rahul is a Principal Engineer at Thoughtworks, based in Gurgaon, India. He is passionate about the craft of building maintainable software through DDD and Clean Architecture, and explores how AI can help teams achieve engineering excellence.

Rahul 是 Thoughtworks 的首席工程师，常驻印度古尔冈。他热衷于通过领域驱动设计（DDD）和整洁架构构建可维护的软件，并探索人工智能如何帮助团队实现卓越的工程水平。

CONTENTS 内容

The Frustration Loop 挫折循环
The Speed Trap 测速陷阱
The Tool-to-Teammate Shift
从工具到队友的转变
Five Patterns for AI Collaboration
人工智能协作的五种模式

PATTERNS 图案

Knowledge Priming 知识启动
Design-First Collaboration
设计优先的协作
Context Anchoring 情境锚定
Encoding Team Standards 编码团队标准
Feedback I

Table of Contents 目

The Shared Mental Model

共享心智模型

Trade-offs and Limitations

权衡与局限性

Where This Leads 这将导致什么?

SIDEBARS 侧边栏

These patterns as installable infrastructure

这些模式作为可安装的基础设施

GENERATIVE AI 生成式人工智能

When I do pair programming with a colleague, certain rituals happen naturally. I walk them through the codebase before asking them to contribute. We sketch on a whiteboard before diving into implementation. I explain our conventions, our constraints, the reasoning behind past decisions. We review the work against team standards.

当我与同事结对编程时，一些惯例自然而然地形成。在让他们贡献代码之前，我会先带他们浏览一遍代码库。在开始实现之前，我们会先在白板上画草图。我会解释我们的约定、限制以及过去决策背后的原因。我们会根据团队标准来审查工作成果。

With AI coding assistants, I have noticed that most developers—myself included, initially—skip all of this. We type a prompt, expect aligned output, and then wonder why the result doesn't fit.

在使用 AI 编码助手时，我注意到大多数开发者——包括我自己在内，最初都是如此——都会忽略这些步骤。我们输入提示，期望得到匹配的输出，然后纳闷为什么结果不符。

The promise of AI coding assistants was compelling: dramatically faster development, instant code generation, a tireless pair programmer available around the clock. The adoption has been remarkable—by most accounts, the majority of professional developers now use tools like GitHub Copilot, Cursor, or Claude in their daily work.

AI 编码助手的前景令人瞩目：开发速度大幅提升、代码即时生成，以及全天候待命的结对程序员。其应用普及程度令人瞩目——据大多数统计，如今大多数专业开发人员都在日常工作中使用 GitHub Copilot、Cursor 或 Claude 等工具。

Yet a consistent frustration seems to be emerging. The time saved by AI-generated code is often consumed by the effort required to correct it. I have experienced—and heard others describe—a cycle I call the “Frustration Loop”: generate code, review it, find it doesn't fit the codebase, regenerate with corrections, review again, and

eventually either accept heavily-modified output or abandon the attempt entirely.

然而，一种持续的挫败感似乎正在显现。人工智能生成的代码节省的时间，往往会被修正代码所需的精力所抵消。我亲身经历过——也听其他人描述过——一个我称之为“挫败循环”的怪圈：生成代码，审查代码，发现它与代码库不兼容，重新生成代码并进行修正，再次审查，最终要么接受大幅修改后的输出，要么彻底放弃尝试。

The pattern is familiar enough that it warrants examination. My sense is that this is not a failure of AI capability—modern language models are remarkably sophisticated. The friction arises from how we collaborate with these systems.

这种模式非常常见，值得深入研究。我认为这并非人工智能能力的不足——现代语言模型已经非常复杂。问题在于我们如何与这些系统协作。

The Frustration Loop 挫折循环

The typical interaction pattern looks something like this:

典型的交互模式大致如下：

A developer asks the AI to create a service. The AI responds quickly with syntactically correct code that follows common patterns from its training data. However, it uses Express.js when the project uses Fastify. It places files in `utils/` when the convention is `lib/services/`. It generates class-based code when the codebase is functional. Each correction requires another generation cycle, and with each cycle, the time advantage erodes.

开发者要求人工智能创建一个服务。人工智能迅速响应，生成语法正确的代码，这些代码遵循其训练数据中的常见模式。然而，当项目使用 Fastify 时，它却使用了 Express.js。此外，它把文件放在了 `utils/` 目录下，而规范是..... `lib/services/` 。当代码库功能正常时，它会生成基于类的代码。每次修正都需要一个新的生成周期，而随着周期的增加，时间优势会逐渐消失。

```
Generate → Review → "Not quite right" → Regenerate → Review → "Still wrong" → Give up
```

This cycle persists because AI assistants, by default, draw from their training data—an aggregation of patterns from millions of repositories. They produce what might be called “the average of the internet” rather than code that fits a specific team's architecture and conventions.

这种循环之所以持续存在，是因为人工智能助手默认会从其训练数据中汲取信息——这些数据汇总了来自数百万个存储库的模式。它们生成的是所谓的“互联网平均值”，而不是符合特定团队架构和规范的代码。

Common symptoms include:

常见症状包括：

- AI generates solutions that don't align with existing architecture

  人工智能生成的解决方案与现有架构不符。
- Developers spend significant time on post-generation editing

  开发人员花费大量时间进行后期编辑
- Context established early in a conversation is lost as the session lengthens

  对话初期建立的语境会随着对话时间的延长而消失。
- Quality varies depending on which team member is prompting

  质量会因提问的团队成员不同而有所差异。

The Speed Trap 测速陷阱

Part of the problem lies in how success is measured with these tools. Teams often track metrics like time to first output or lines of code generated. These metrics are easy to capture and show immediate results, but they obscure the actual cost.

部分问题在于如何衡量这些工具的成功。团队通常会追踪诸如首次输出时间或生成的代码行数之类的指标。这些指标易于获取且能立即显示结果，但却掩盖了实际成本。

If an AI generates 200 lines of code in seconds, but a developer spends 30 minutes reviewing, debugging, and refactoring to fit team patterns, the net productivity gain is questionable. The work has shifted from writing to fixing, but the total effort may not have decreased.

如果人工智能在几秒钟内生成 200 行代码，但开发人员却要花费 30 分钟进行审查、调试和重构以适应团队模式，那么最终的生产力提升是否值得怀疑。工作重心从编写代码转移到了修复代码，但总的工作量可能并没有减少。

When I pair with a human colleague, I don't measure success by how fast they type. I care about whether their contribution fits the codebase, whether it solves the right problem, whether it will hold up in review. The same should apply to AI collaboration.

当我与人类同事结对编程时，我不会以他们的打字速度来衡量成功。我更关心的是他们的贡献是否符合代码库的要求，是否解决了正确的问题，以及是否经得起代码审查。同样的道理也适用于人工智能协作。

More useful metrics focus on what I might call “collaboration quality”:

更有用的指标侧重于我所谓的“协作质量”：

Misleading Metric 误导性指标	More Useful Alternative 更实用的替代方案
Time to first output 首次输出时间	First-pass acceptance rate 首次通过率
Lines of code generated 生成的代码行数	Iteration cycles per task 每个任务的迭代周期
Tasks completed 已完成的任务	Post-merge rework required 合并后需要进行返工。
Generation speed 世代速度	Review burden compared to manual writing 与手工编写相比，审阅负担更重

For teams tracking DORA metrics, first-pass acceptance serves as a leading indicator for change failure rate. Code that requires extensive correction before being usable is a signal of misalignment, and misaligned code that ships becomes technical debt.

对于追踪 DORA 指标的团队而言，首次验收率是变更失败率的领先指标。需要大量修正才能使用的代码表明存在方向不一致，而发布时存在方向不一致的代码会成为技术债务。

The Tool-to-Teammate Shift

从工具到队友的转变

A useful reframe, I believe, is to stop treating AI as a tool and start treating it as a teammate—a distinction that sounds semantic but has practical implications.

我认为，一个有用的重新定义是，停止将人工智能视为工具，而开始将其视为队友——这种区别听起来像是语义上的区别，但却具有实际意义。

> *AI assistants are like junior developers with infinite energy but zero context.*
>
> *人工智能助手就像精力无限但缺乏背景知识的初级开发人员。*

AI assistants are like junior developers with infinite energy but zero context. They can work faster than any human, they never tire, and they never complain. But they know nothing about a specific project's conventions, constraints, or history. Without the same scaffolding a team would provide a human collaborator, they default to generic patterns that may or may not fit.

人工智能助手就像精力无限但缺乏上下文知识的初级开发人员。它们工作速度比任何人都快，永不疲倦，也从不抱怨。但它们对特定项目的惯例、限制或历史一无所知。由于缺乏团队为人类协作者提供的指导，它们只能使用可能并不适用的通用模式。

Consider how I work with a new team member. I don't expect them to produce high-quality, architecture-aligned code on their first day without context. I onboard them. I walk through the codebase. I explain the patterns and the reasoning behind architectural decisions. I pair with them on initial work. I review their output against team standards.

想想我是如何与新团队成员合作的。我不会指望他们在第一天就写出高质量、符合架构的代码，更不会让他们在缺乏背景知识的情况下就写出高质量的代码。我会引导他们入职。我会带他们浏览代码库。我会解释架构决策背后的模式和逻辑。我会和他们一起完成最初的工作。我会根据团队标准审查他们的成果。

The parallel to pair programming is instructive:

这与结对编程的相似之处很有启发性：

Human Pair Programming 人机配对编程	AI Collaboration Equivalent 人工智能协作等效物
“Let me show the docs first” “让我先出示一下文件。”	Sharing architectural context before requesting code 在请求代码之前先分享架构背景信息
“Let's sketch this on the whiteboard” “我们把它画在白板上吧。”	Structured design discussion before implementation 实施前进行结构化设计讨论
“Here's how reviews work here” “这里评论的运作方式如下”	Encoding standards into reusable prompts or commands 将标准编码成可重用的提示符或命令
“Let me update the doc with the decision” “让我把决定更新到文档里。”	Persisting decisions so they survive session boundaries 保留决策，使其在会话边界之外仍然有效

Human Pair Programming 人机配对编程	AI Collaboration Equivalent 人工智能协作等效物
"What did that teach us?" "这件事教会了我们什么？"	Systematically capturing what worked and what didn't 系统地记录哪些方法有效，哪些无效

The AI is fast—remarkably fast. But it needs the same things a human pair needs:

人工智能速度很快——快得惊人。但它也需要和人类搭档一样的条件：

- Onboarding — Context about the codebase before contributing
  入职培训——贡献代码库之前需要了解相关背景信息
- Whiteboarding — Structured design discussion before implementation
  白板讨论——实施前的结构化设计讨论
- Guardrails — Standards and quality checks consistently applied
  护栏——始终如一地应用标准和质量检查

Five Patterns for AI Collaboration

人工智能协作的五种模式

The framework I propose consists of five complementary patterns. Each addresses a specific failure mode in AI collaboration—and each mirrors a practice that makes human collaboration effective.

我提出的框架由五种互补模式组成。每种模式都针对人工智能协作中的一种特定失败模式，并且每种模式都反映了一种使人类协作有效的实践方法。

Knowledge Priming 知识启动

The human parallel: Onboarding a new hire

人类的类比： 新员工入职培训

Before asking AI to generate code, share curated context about the project: the tech stack with version numbers, directory structure, naming conventions, and examples of existing patterns. This practice is essentially manual RAG (Retrieval-Augmented Generation)—filling the context

Design-First Collaboration

设计优先的协作

The human parallel: Whiteboarding before coding

人类的类似例子： 编码前先进行白板演示

Rather than asking AI to immediately produce code, walk through progressive levels of design: capabilities, components, interactions, contracts, and only then implementation. This mirrors the

window with high-value, project-specific information that overrides generic training data.

在让 AI 生成代码之前，请先提供精心整理的项目上下文信息：包括技术栈及其版本号、目录结构、命名规范以及现有模式示例。这种做法本质上是一种手动 RAG（检索增强生成）——用高价值的、项目特定的信息填充上下文窗口，从而覆盖通用的训练数据。

Just as I would walk a new team member through the codebase before asking them to contribute, I prime the AI with the context it needs to produce aligned output.

就像我会在要求新团队成员贡献代码之前先带他们了解代码库一样，我会先给 AI 提供它需要的上下文信息，以便它能够生成正确的输出。

The failure mode addressed: AI defaults to generic patterns because it lacks project-specific context.

所解决的故障模式： 人工智能由于缺乏项目特定的上下文，因此默认使用通用模式。

read more... 阅读更多...

whiteboarding that precedes coding in effective human collaboration.

与其要求人工智能立即生成代码，不如循序渐进地进行设计：先构建功能，再构建组件、交互、契约，最后才进行实现。这类似于高效的人类协作中，在编写代码之前进行白板讨论的过程。

When I pair with a colleague on a complex feature, we don't jump straight to the keyboard. We sketch. We align on approach. We catch misunderstandings before they become bugs. The same discipline applies to AI.

当我和同事合作开发复杂功能时，我们不会直接开始敲代码。我们会先画草图，商定开发思路，并在误解演变成 bug 之前及时发现并解决。同样的原则也适用于人工智能开发。

The failure mode addressed: AI jumps to implementation before understanding requirements, producing code that solves the wrong problem.

所解决的失败模式： 人工智能在理解需求之前就直接进行实现，导致生成的代码解决的是错误的问题。

read more... 阅读更多...

Encoding Team Standards 编码团队标准

The human parallel: What a senior developer does instinctively

人类的类比： 资深开发人员的本能行为

Every team has senior developers who instinctively know when code is right—they spot architectural drift, security gaps, and convention violations that others miss. This intuition is invaluable, but it lives in their heads. It does not scale.

每个团队都有一些资深开发人员，他们凭直觉就能判断代码是否正确——他们能发现其他人忽略的架构偏差、安全漏洞和违反规范之处。这种直觉非常宝贵，但它只存在于他们的脑海中，无法扩展。

The solution is to make tacit knowledge explicit. Just as I would walk a new hire through our

Context Anchoring 情境锚定

The human parallel: Updating the doc with decisions

人类的类比： 用决策更新文档

Maintain a living document that captures decisions, constraints, and current state as a feature evolves. This document serves as external memory, anchoring context that would otherwise be lost as conversations grow longer or sessions end.

维护一份动态文档，记录功能开发过程中的决策、限制条件和当前状态。这份文档如同外部记忆，锚定了可能随着讨论时间延长或会议结束而丢失的上下文信息。

When pairing with humans, I naturally update shared documents as we make decisions. The same practice prevents AI from contradicting

conventions, I make those defaults explicit and shareable for AI collaboration. When defaults exist as artifacts rather than intuition, they apply consistently regardless of who is prompting.

解决方案是将隐性知识显性化。就像我会向新员工讲解我们的惯例一样，我也会将这些默认设置显性化并使其可共享，以便与人工智能协作。当默认设置以实物形式存在，而非仅仅依靠直觉时，无论由谁提示，它们都能始终如一地适用。

The failure mode addressed: AI generates technically correct code that violates team expectations, requiring extensive rework.

所解决的故障模式： 人工智能生成技术上正确的代码，但违反了团队的预期，需要进行大量返工。

read more... 阅读更多...

Feedback Flywheel 反馈飞轮

The human parallel: Retrospectives and continuous learning

人类的类比： 回顾与持续学习

Systematically harvest learnings from AI interactions to improve the other four patterns. When a prompt works particularly well, capture it. When AI consistently misunderstands something, add it to the priming context. When a failure pattern emerges, add a guardrail.

系统地从人工智能交互中汲取经验，以改进其他四种模式。当某个提示效果特别好时，将其记录下来。当人工智能持续误解某些内容时，将其添加到引导上下文中。当出现故障模式时，添加防护措施。

Just as teams learn from retrospectives and improve their collaboration over time, I build institutional knowledge about what makes AI collaboration effective.

正如团队通过回顾会议学习并随着时间的推移改进协作一样，我积累了关于如何使人工智能协作有效的机构知识。

The failure mode addressed: Teams make the same mistakes repeatedly without building institutional itself or forgetting earlier conclusions.

在与人类协作时，我会随着决策的做出自然而然地更新共享文档。同样的做法也能防止人工智能自相矛盾或忘记之前的结论。

The failure mode addressed: AI forgets decisions made earlier in long conversations, leading to contradictions and inconsistency.

所解决的故障模式： 人工智能会忘记在长时间对话中做出的决定，从而导致矛盾和不一致。

read more... 阅读更多...

knowledge about effective AI collaboration.

所解决的失败模式： 团队反复犯同样的错误，却没有建立关于有效人工智能协作的机构知识。

read more... 阅读更多...

The Shared Mental Model

共享心智模型

These patterns work together to create what might be called a shared mental model between humans and AI. When both operate from the same vocabulary (through priming), the same architecture vision (through design-first discussion), the same quality standards (through codified commands), and the same decision history (through anchored documentation), the friction of translation diminishes.

这些模式共同作用，在人类和人工智能之间构建了一种所谓的共享心智模型。当两者都使用相同的词汇（通过引导）、相同的架构愿景（通过设计优先的讨论）、相同的质量标准（通过规范化的命令）以及相同的决策历史（通过锚定文档）时，翻译过程中的摩擦就会减少。

The cognitive load shifts from constant vigilance and correction to expressing intent and refining output. This is the difference between struggling with a tool and collaborating with a capable partner.

I hypothesize that these patterns, when applied consistently, could yield meaningful improvements:

- Higher first-pass acceptance rates (reducing the generate-fix-regenerate cycle)
- Fewer iteration cycles per task
- Less post-merge rework

These are not yet validated findings—this is a proposed framework based on reasoning about why friction occurs and what might address it. Early experiments in my own work have been encouraging, but more practice and observation is needed.

Trade-offs and Limitations

This approach is not without costs. Creating and maintaining priming documents requires effort. Design-first conversations take longer than immediately requesting code. The patterns require discipline to sustain.

For simple, one-off tasks, the overhead may not be justified. A quick utility function or a straightforward refactor may not warrant a full design discussion. The investment pays off primarily for non-trivial work, especially work that spans multiple sessions or involves team coordination.

There is also a learning curve. Teams accustomed to treating AI as a search engine or autocomplete system may find the transition to structured collaboration unfamiliar. The shift requires deliberate practice.

Where This Leads

I have outlined the problem of friction in AI-assisted development and introduced a framework for addressing it. Each of the five patterns merits deeper exploration:

- Knowledge Priming: What to include, how to structure it, how to keep it current
- Design-First Collaboration: The levels of design, handling AI that wants to skip ahead
- Encoding Team Standards: Making tacit knowledge explicit, scaling senior intuition across the team
- Context Anchoring: Managing long conversations, surviving session boundaries
- Feedback Flywheel: Systematic improvement, measuring what matters

Subsequent articles will explore each pattern in depth.

The underlying insight is straightforward: AI assistants are powerful but contextless. The techniques that make human collaboration effective—onboarding, structured discussion, shared standards, documented decisions, continuous learning—apply equally to AI collaboration. My belief is that developers who internalize this approach will experience AI as a force multiplier rather than a source of friction.

These patterns as installable infrastructure

Lattice is an open-source framework that operationalizes all five patterns as composable AI skills — available as a plugin for AI coding assistants. The series is the thinking; Lattice is the infrastructure.

Acknowledgements

I'm grateful to several people who shaped this series. Jim Gumbley read early drafts and pushed on the layer beneath the practical advice—especially the connection to how attention and context work in transformer models. His feedback sharpened the thinking and sparked ideas for future work.

Vanya Seth has been a long-standing conversation partner on these ideas; she encouraged and guided me to draft articles for Martin's site.

I treat AI as a teammate, not a tool—the same philosophy behind this series. I anchored the base thought process: the ideas, structure, and voice. AI accelerated the drafting and complemented my thinking with quick iterations and alternative phrasings. Together we complemented each other; the writing process itself reflects the kind of collaboration these articles advocate.

▶ Significant Revisions

© Martin Fowler | Disclosures