当软件可以
边用边长大

我对AI时代开发方式的一点思考

我越来越觉得，AI给软件开发带来的变化，远不止代码写得更快。它正在缩短一个过去经常被忽略的距离：从“我有一个想法”，到“我可以亲手试一试”，究竟要等待多久。

过去，需求常常要经过沟通、排期、开发和测试，用户可能数天甚至数周后才能看到功能。今天，对于范围明确的小功能，这段时间有时可以缩短到数小时。上午提出想法，下午就能上手，再根据体验继续修改。这是有条件的实践观察，并非适用于所有项目的速度承诺。

等待几周交付时，人主要在想象软件；很快就能上手时，人开始通过使用来思考软件。编码提效由此改变了需求如何形成、用户如何参与，以及软件怎样成长。

漫长而不可见的开发过程，容易让普通人感到无力：不知道需求是否被理解，也担心一次改口会带来巨大返工。亲眼看到自己的意见变成可用功能，则让人更愿意继续尝试。

> ### 可见的进步，是继续参与的理由
>
> Amabile与Kramer的“进展原则”指出，在有意义的工作中，微小进步也能积极影响情绪和动力。[1] 我据此推想，频繁看到可用成果，有助于普通人持续参与开发；这项研究本身并未直接验证AI编程。

这里更准确的说法是“较短周期的可见反馈”，而不是把数小时等同于心理学实验中的即时刺激。

先解决一件事

让能用的软件成为沟通工具

计划驱动的开发习惯先整理产品需求文档（PRD）、设计架构、约定接口，再分工实现。这有其合理性：人工开发与多人协调成本较高，提前想清楚可以减少返工。但业务用户往往只能清楚描述眼前的问题，难以预见全部流程和例外。

例如，用户最初只想“把发票整理起来”。看到第一版后，才发现需要按供应商筛选；处理一批真实单据，又意识到必须识别重复发票。这些想法可能来自使用后的新认识，不能简单归咎于最初“没提清需求”。

因此，我更愿意从一个最小可用闭环开始：先完成一张发票的上传、识别、核对和保存。功能不多，但能从头到尾做成一件事。用户围绕结果反馈，AI据此调整，需求记录也随之完善。

> **已有方法提供的启发：构建—衡量—学习**
>
> Eric Ries的精益创业方法强调，用最小可行产品（MVP）启动学习，并根据真实反馈判断继续还是调整。[2] 这支持“通过做出来澄清需求”的思路，也提醒我：衡量进步不能只数新增功能，还应看用户是否真的解决了问题。

DORA也强调“小批量工作”：缩小每次变化，有助于更快获得反馈，并控制AI加速开发带来的交付不稳定性。[3] 因此，我的判断不是“AI只能做小软件”，而是：无论系统最终多大，每轮工作都应当足够小、边界清楚、能够验收。

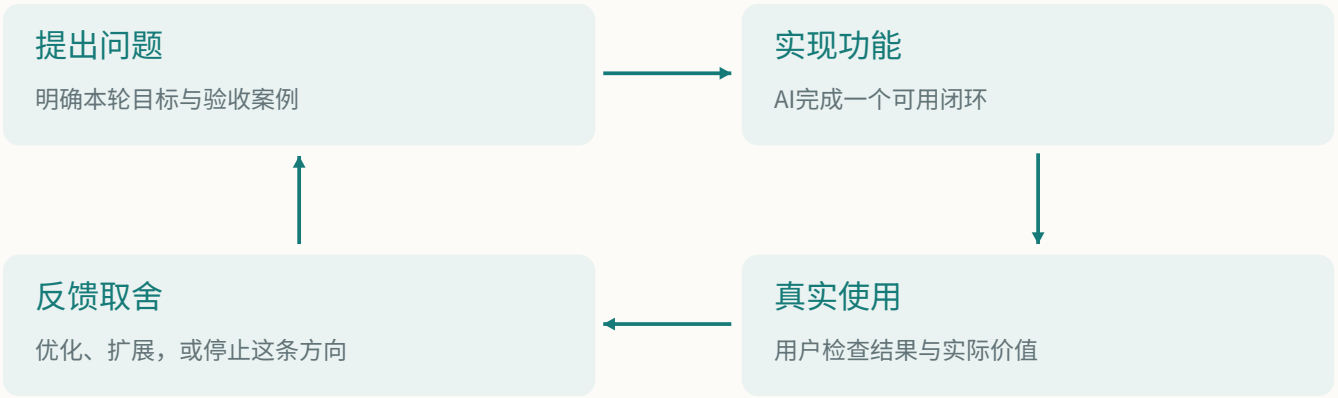
一个实用的验收问题：拿一份真实材料，用户能否独立完成这件事，并得到正确结果？

我的初步设想

体验驱动的渐进式开发

我暂且用这个名字概括自己的想法，它还不是成熟的方法论。我希望把开发设计成两个互相配合的循环：一个不断发现值得做的功能，一个维护继续开发的能力。

体验循环 · 每次只推进一个可用目标



治理循环 · 出现结构性问题时介入

梳理问题 → 分步重构 → 回归验证 → 恢复迭代

触发信号：重复逻辑增多、规则不一致、修改频繁牵连其他功能。

小循环里，用户决定什么值得做，AI负责辅助实现与验证。每轮都留下可运行版本、需求变化和检查结果，让下一轮有据可依。反馈也可以导向删减功能，而不只是不断增加。

大循环里，AI梳理代码、数据流和依赖，说明哪些问题妨碍扩展、怎样分步改善。这里的“治理”，就是把内部关系整理清楚，让后续修改更可靠。它的完成标准应是原有功能仍然正确，目标问题有所改善。

日常小修整持续进行，跨模块问题集中评审。两个循环按问题触发，不必机械地每隔几轮就重构，更不能等系统失控后才整体重写。

继承思想，改变参与方式

这与传统敏捷有什么不同?

首先，它继承了敏捷。尽早交付、接受变化、业务与开发协作，本来就是《敏捷宣言》的核心原则；敏捷也强调良好设计。[4] 我并不认为自己发明了替代敏捷的新理论，更愿意把它看作面向非专业用户的一种具体组织方式。

观察维度	常见的团队敏捷实践	本文设想的人机协作方式
参与方式	用户反馈，由产品与工程团队组织实现	业务用户直接与AI持续协作，决定每轮目标
需求表达	通过需求条目、沟通与评审逐步澄清	增加对可运行结果的直接操作，在体验中澄清
反馈节奏	由任务规模与交付能力决定，也可非常快	适合的小功能争取数小时可体验，缩短想法与验证的间隔
质量保障	工程团队持续承担设计、测试与维护	将测试、记录和治理显式放入流程，弥补用户技术判断不足

表中是实践侧重的比较，并非互斥分类；成熟敏捷团队也可以采用右侧做法。

我认为，AI的意义在于降低了这种协作的实现成本。反馈若能在同一天发生，用户还记得刚才为什么那样想，就已经可以检查结果。与此同时，许多原本由工程师默默承担的工作，需要被主动写进流程。

> **变化发生在三个地方**
>
> 用户更直接地参与实现；想法更快得到体验验证；工程质量保障需要更清楚地呈现。

快，也要能够继续改

把治理设计进开发过程

从小功能开始，仍然需要适度的前期设计。核心数据怎样保存、谁能够访问、模块怎样分工，以及哪些选择日后难以改变，都应该尽早想清楚。可以推迟没有依据的复杂设计，但不能省略已经明确的基本约束。

Martin Fowler讨论的“演进式设计”指出，设计可以伴随实现逐步发展，但需要测试、持续集成和重构支撑，也允许适度的初始架构。[5] 这为我的设想提供了重要基础：架构能够逐步完善，前提是每一步都有维护秩序的办法。

AI也需要这种约束。Anthropic在长时间运行的智能体实验中发现，一次承担太多工作、会话间丢失进展、没有充分测试便宣布完成，都会阻碍项目推进；其改进包括逐项实现、记录进度和端到端验证。[6] 这支持小步推进，但不是所有模型和项目都适用的保证。

所以，我设想每轮至少留下三样东西：可恢复的版本、更新后的需求与决策记录，以及实际验证结果。架构调整先说明目的，再分步实施，并检查已有功能是否受损。AI自查可以帮助发现问题，却不能自动等同于质量保证。

对于非技术用户，职责也随之变化：不必掌握每段代码，却需要提供真实案例、明确优先级，并判断结果是否符合业务事实。涉及资金、敏感数据或关键业务链路时，还需要相应的专业审查。

> 让用户的理解与软件的能力一起成长
>
> 先实现一个可验证的价值，在使用中澄清需求，再通过持续整理获得继续扩展的能力。这是我目前最希望尝试的开发路径。

这仍然是个人尚不成熟的思考，更适合从边界清楚、风险可控的应用开始。已有研究分别支持它的若干环节，整套方式的效果仍需实践验证。但我相信，当普通人能持续看到自己的想法产生作用，软件开发就有机会成为一种可以亲身参与、逐步学会的创造过程。

原始资料 · 可点击访问

哪些观点支持了这篇文章?

以下资料分别支持动机、需求验证与工程实践；它们并未共同验证本文提出的完整方案。正文中的编号对应下列来源。

[1] 进展原则 / The Progress Principle

Teresa Amabile、Steven Kramer · 相关研究发表于2010—2011年

支持：有意义工作的微小进展与积极情绪、动力相关。本文延伸：可见成果可能帮助用户持续参与AI开发。

[2] 精益创业 / Build–Measure–Learn

Eric Ries · The Lean Startup 官方方法说明

支持：用最小可行产品和真实反馈推进经验证的学习。本文借鉴其需求探索方式，不将个人工具开发等同于创业。

[3] 小批量工作 / Working in Small Batches

DORA · 能力指南，页面更新于2025年12月8日

支持：缩小变化、缩短反馈，并改善AI辅助开发的交付条件。研究结论不等于任意项目的因果保证。

[4] 《敏捷宣言》遵循的原则

敏捷宣言 · 2001年

支持：尽早交付、拥抱变化、业务协作与良好设计。本文是在这一思想基础上的人机协作设想。

[5] Is Design Dead? / 演进式设计

Martin Fowler · 2004年版本

支持：设计可随实现演进；测试、持续集成、重构和适度前期设计是重要支撑。

[6] Effective Harnesses for Long-running Agents

Anthropic Engineering · 2025年11月26日

支持：逐项推进、保留进度与实际流程验证。属于特定智能体实验的工程经验，不是普遍成功率证明。

资料核验：2026年10月7日。文中研究观点均为概述；图示与比较框架为本文整理。