智书企飞 AI 辅助编程季度复盘：日均合并 MR 增加约 82%，然后呢？

AI 辅助编程（AI Coding）在智书研发中的季度实践：一项 181 天的描述性前后对照研究

研究类型	单组织描述性前后对照案例研究	观察期	2026 年 1 月 1 日–6 月 30 日，共 181 天
数据范围	31 个业务代码仓库及团队实践记录	数据截至	2026 年 6 月 30 日

研究摘要

研究问题。当 AI 辅助编程从少数成员的个人尝试进入研发团队的日常工作，研发交付活动、责任边界和质量治理方式发生了什么变化？智书团队在实践中遇到了哪些问题，又采取了哪些调整？

研究方法。本文比较智书团队集中推进 AI 辅助编程前后两个阶段的研发活动。集中推进前为 2026 年 1 月 1 日至 3 月 19 日，共 78 天；集中推进后为 3 月 20 日至 6 月 30 日，共 103 天。由于两个阶段长度不同，合并请求（MR）数量统一换算为日均值。定量数据来自 31 个业务代码仓库中已经去重的合并 MR 记录；团队复盘和实践记录用于补充协作方式、使用问题及治理措施。两类材料分别解释，不把实践感受换算成生产率或质量结果。

主要发现。集中推进后，日均合并 MR 数由 36.7 个增至 66.9 个，按展示值计算增幅约为 82%。这说明进入交付环节的研发变更多了。与此同时，部分需求开始由一名主责工程师推动前后端实现与交付，我们也建立了失败案例复盘、任务技能与提示词沉淀、仓库级 AGENTS.md、多层代码审查、跨技术栈交叉评审和自动化验证机制。

结论边界。本研究没有同期对照组，也缺少人员投入、需求规模、交付周期和缺陷结果等同口径数据。因此，现有证据不能证明 AI 辅助编程已经提高人均生产率、软件质量或业务价值。

关键词： AI 辅助编程；研发交付；代码审查；跨技术栈协作；质量治理

1. 研究背景与问题

智书团队最初在文档初稿、代码生成、问题排查、资料整理和测试用例等具体任务中尝试 AI。随着使用范围扩大，AI 逐步进入需求分析、方案设计、开发实现、测试验证和代码审查。团队关注的问题也随之改变：早期主要判断 AI 能否完成任务，后期则更关心输出如何验证、经验如何复用，以及最终由谁对结果负责。

本文关注三个问题：

01 我们集中推进 AI 辅助编程前后，合并 MR 活动发生了什么变化？

02 我们的需求责任与协作方式出现了哪些变化？

03 面对业务上下文、验证和跨技术栈风险，智书建立了哪些治理机制？

2. 实践推进过程

2.1 探索期：从个人尝试开始

我们的故事开始于几个工程师的私下尝试。有人用 AI 起草接口文档，有人让它生成增删改查代码，还有人拿它排查线上问题、整理测试用例。那时候每个人的用法都不一样，没有统一规范：有人觉得“真香”，有人觉得“还不如自己写”。

我们很快碰到了第一个问题：**业务上下文不足**。AI 可以在几秒内生成一段结构完整的代码，但它不知道我们的调用链长什么样、边界条件有哪些、异常状态怎么处理、上下游依赖是什么，也不知道历史上为什么做了那个看起来“奇怪”的设计约束。结果是，生成的代码看起来很对，但一跑就出问题，最后还是得靠人逐行检查。

这个阶段我们做的第一个调整是**失败案例复盘**。我们开始把那些“AI 生成了但实际不能用”的案例记录下来：错误判断是什么、遗漏了哪些条件、为什么返工。这些复盘后来成为团队改进规则的素材。

2.2 融合期：进入完整工作链路

个人使用逐渐稳定后，AI 不再只是写代码的工具，开始进入需求分析、方案设计、开发实现、测试验证和代码审查等环节。

一个明显的变化是，**工程师开始跨技术栈工作**。前端同学借助 AI 阅读后端代码、理解数据库逻辑，然后自己把接口调通；后端同学也开始写前端页面。部分需求的开发责任从原来的“前端做一段、后端做一段、中间交接”，转为由一名主责工程师推动前后端实现与交付。

但我们很快意识到了风险：**能改代码不等于懂这个领域**。一个前端工程师可以借助 AI 写出能跑的后端代码，但他可能不知道这个表的索引为什么这么建、这个接口的性能瓶颈在哪、哪个历史兼容逻辑不能动。责任范围扩大了，能力积累却没有同步跟上，错误判断可能影响更大的业务范围。

于是我们做了两个调整。一是**跨技术栈交叉评审**：跨技术栈的改动必须由熟悉该领域的成员检查方案和代码，重点看架构、性能、安全和历史约束。二是**沉淀 Skill（任务技能）和 Prompt（提示词）**：把生成接口文档、排查空指针、补充单元测试等重复任务所需的上下文、操作步骤、输出要求和验证方式固定下来，变成团队可以复用的资产。

2.3 治理形成期：从个人经验到共同规则

随着 AI 进入更多环节，我们越来越觉得，不能只靠个人经验了。每个人的提示词、检查标准和对 AI 输出的信任程度都不一样。我们需要一套团队共同维护的规则和工具，这就是内部代码审查工具“小智”的由来。它的迭代也反映了我们的治理思路。

4 月中旬 — **想法萌芽：先发现明显问题**

MR 越来越多，人工审查跟不上，格式不规范、漏改配置、遗漏边界条件等问题反复出现。我们提出开发代码审查机器人，希望它在人工审查之前先扫一遍，把明显的问题拦下来。

5 月 19 日 — **正式上线：先暴露风险，再由人判断**

小智加入代码审查群，开始接受真实的 MR 评审请求。它用于提前暴露明显风险、低级问题和需要人工确认的点，不替代人工审查。运行记录中，首轮结果的中位耗时（P50）约为 45 秒，约 90% 的评审可在 3 分钟内完成；该耗时描述尚未提供明确的统计窗口、样本量和计时起止口径，不用于前后效率比较。

6 月第一周 — **处理大 MR 和异常**

将超出模型上下文的大规模代码差异拆成多个分片逐个审查，把 SQL 变更纳入审查范围，并在模型输出格式异常时自动重试。

6 月第二周 — **补上复审和协作**

开发者补充说明后，小智能基于最新评论重新判断；前后端等关联 MR 可以联合审查；通过结果在群里展示得更直观。

6 月第三周 — **补齐上下文，明确未覆盖范围**

将群内重点、背景和 MR 描述更稳定地带入审查，改进复审时是否参考上轮评论的判断。大 MR 分片覆盖不完整时，不把未审完的当成通过，并在 GitLab 评论中明确未覆盖的部分。

6 月第四周 — **引入第二模型，收紧判断边界**

接入第二模型进行 A/B 分组比较，考察审查质量与成本。对证据不足的问题优先标为“需要人工确认”，减少误报直接成为阻塞项；主模型额度异常或服务不可用时，由另一个模型兜底补审。这里描述机制调整，不据此判断模型优劣。

本地与线上配合。我们同时维护本地审查助手 qfei-code-review，以 Codex 任务技能的形式安装。它在开发现场读取完整代码和 Git 历史，按实际合入范围审查，生成**带签名的审查凭证**。线上小智负责验签，确认本地审查是否仍然有效，再按统一范围完成线上审查，结论发布到 GitLab 评论，供团队查看和追踪。同一套规则、同一个审查范围，本地结论仍需线上校验。

与此同时，我们持续建设**自动化测试平台**，把接口定义、自动化用例、测试报告和执行记录整理为可复用的验证资产。我们也在各代码仓库中维护 AGENTS.md，将项目结构、技术约束、开发与测试要求、变更边界、审查规则及常见问题固化为仓库级上下文，随项目演进和复盘持续更新。

这些实践形成了从生成、校验、交付到复盘和规则更新的治理闭环。人工责任贯穿每个环节；生成变快之后，校验、复盘和追责的能力也需要跟上。

图 1　AI 辅助编程的团队治理闭环。自动化测试属于校验环节，工具不替代人的最终责任。

3. 数据与研究设计

3.1 研究设计

本研究采用单组织描述性前后对照设计。该设计用于识别团队集中推进 AI 辅助编程前后同时出现的变化，不用于单独识别因果效应。

两个阶段长度不同，合并 MR 统一换算为自然日日均值。

图 2　研究观察窗口与阶段切分。以团队集中推进 AI 辅助编程的日期作为切换点。

阶段	日期范围	天数	状态说明
集中推进前	2026-01-01–2026-03-19	78 天	团队集中推进前阶段
集中推进后	2026-03-20–2026-06-30	103 天	以团队开始集中推进 AI 辅助编程的日期作为阶段切分点

3.2 数据材料

材料	统计对象	用途	当前边界
代码仓库记录	31 个业务仓库中已去重的合并 MR	比较集中推进前后的交付活动	已清理重复记录和机器人记录；尚未确认非业务变更是否排除，具体排除条件、去重键及 MR 归属日期字段仍需补充
团队实践记录	协作方式、使用问题和复盘结果	解释责任变化与风险类型	不用于估计问题发生率
智能评审记录	本地任务技能规则检查与小智运行记录	描述评审机制的采用	当前不用于证明缺陷率下降
自动化验证记录	接口定义、自动化用例、测试报告和执行记录	描述验证能力的建设	失败重试和重复记录已去重；缺少推进前同口径基线

3.3 指标定义与分析原则

本文的核心定量指标为**日均合并 MR 数**：观察阶段内已去重的合并 MR 数量除以该阶段的自然日天数。选择日均值是为了减少两个观察窗口长度不同带来的直接影响。

合并 MR 是研发活动指标，不等同于完成需求数、人均生产率、代码质量或业务价值。实践记录只用于说明协作变化和问题类型，不将主观感受换算为量化改善。

4. 研究结果

我们从研发交付活动、协作方式和质量治理能力三个维度观察集中推进后的变化。核心数据汇总如下：

维度	指标	集中推进前	集中推进后 / 当前值	说明
研发交付	日均合并 MR 数	36.7 个/天	66.9 个/天（约 +82%）	31 个业务仓库去重记录
协作方式	主责工程师单人闭环需求	未量化	定性观察：部分需求转为单人推动前后端交付	缺少需求总数和统一判定规则，暂不量化
质量治理	小智智能评审	未提供同口径基线	1,458 次任务（5 月 19 日–6 月 30 日），96.7% 稳定完成	覆盖 753 次变更、37 位成员、42 个仓库

维度	指标	集中推进前	集中推进后 / 当前值	说明
质量治理	自动化验证资产	未提供同口径基线	接口定义 1,941 个、API 自动化用例 3,072 个、报告 1,242 份	用例执行 99,899 次：成功 95,072 次、失败 4,827 次

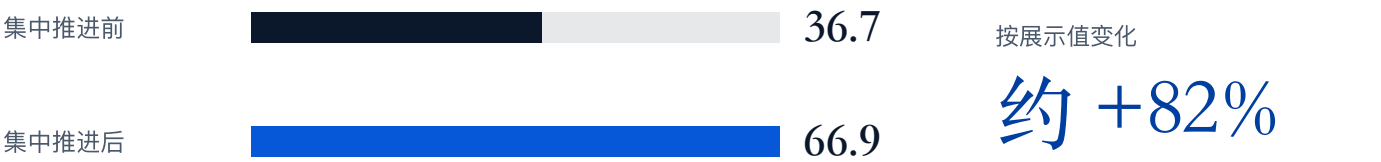
4.1 合并 MR 活动增加

指标	集中推进前	集中推进后	变化
日均合并 MR 数	36.7 个	66.9 个	约 82%

集中推进后，进入交付环节的研发变更数量增加。由于目前只有两个阶段的汇总日均值，尚不能判断增长是否普遍发生在多数仓库，还是由少数仓库或集中交付项目带动。

日均合并 MR

单位：个/自然日



该指标反映研发变更活动，不等同于生产率或业务价值。

图 3 集中推进前后日均合并 MR 数对比。数据来自 31 个业务代码仓库的去重记录。

4.2 需求责任边界扩大

实践记录显示，需求交付开始由前后端分段协作，转向由一名主责工程师推动前后端实现与交付。成员也开始借助 AI 阅读其他技术栈代码并形成实现方案，但这不能替代领域知识。架构、性能、安全和历史约束仍需由具备相关经验的成员检查。

这项变化目前属于定性观察。由于缺少需求总数、单人闭环数量和统一判定规则，本文不报告具体比例。

4.3 质量治理开始进入工作流

代码审查和自动化验证不再只依赖个人习惯。团队将本地任务技能规则检查、小智多模型复核、人工评审和自动化测试接入交付过程，并开始积累可复用的检查规则与验证记录。

2026 年 5 月 19 日至 6 月 30 日，小智形成 1,458 次评审任务，其中 1,410 次稳定完成；覆盖 753 次研发变更、37 位成员和 42 个代码仓库。这里的 42 个仓库是智能评审的采用范围，不等同于核心 MR 分析使用的 31 个业务仓库。

智能评审采用

5 月 19 日—6 月 30 日

96.7%

稳定完成 1,410 / 1,458 次任务

研发变更	参与成员	采用仓库
753 次	37 位	42 个

采用范围用于描述评审机制进入工作流，不用于证明缺陷率下降。

图 4　小智智能评审的阶段性采用情况。42 个采用仓库与核心 MR 分析的 31 个业务仓库口径不同。

截至 2026 年 6 月 30 日，自动化测试平台覆盖 5 个项目、11 个产品，沉淀 1,941 个接口定义、3,072 个 API 自动化用例和 1,242 份测试报告。报告内记录 99,899 次用例执行，其中成功 95,072 次、失败 4,827 次。失败重试和重复记录已经去重；成功状态占比不等同于缺陷率或产品质量。

自动化验证资产

截至 6 月 30 日

接口定义	API 自动化用例	测试报告
1,941	3,072	1,242

报告内用例执行次数

99,899 次

成功 95,072　失败 4,827

失败重试和重复记录已去重；成功状态占比不等同于缺陷率或产品质量。

图 5　自动化验证资产与报告内用例执行记录。覆盖 5 个项目、11 个产品。

这些记录说明质量治理工具已经进入日常工作流，但工具采用范围扩大不等于软件质量已经提高。要回答质量是否改善，还需要缺陷、返工、回滚和生产事故等结果数据。

5. 暴露的问题

5.1 业务上下文不足

AI 根据输入生成内容，不能自动获得完整的业务背景。上下游依赖、异常状态、边界条件和历史设计约束容易被遗漏。输入不足时，推测也可能以确定语气出现，增加人工识别错误的难度。

5.2 验证工作比重提高（定性观察）

部分实践记录提到，工作重心正在向事实确认、调用链检查、测试补充和业务规则核对转移。当前没有工时记录，因此不能计算验证成本的变化幅度。

5.3 责任范围与能力积累不同步

成员可以借助 AI 修改其他技术栈的代码，但完成代码改动并不代表已经掌握相关领域的架构、性能、安全和历史约束。由此带来的风险是：责任范围扩大后，错误判断可能影响更大的业务范围。

5.4 现有指标不足以判断最终效果

合并 MR 增加说明研发变更活动上升；智能评审和自动化验证进入工作流，说明治理能力正在建设。这些指标不能替代交付周期、缺陷率、返工率、回滚次数和最终业务结果。

6. 讨论

现有结果说明，团队集中推进 AI 辅助编程后，研发变更活动增加，成员参与的任务范围扩大，评审与验证机制也随之调整。实践记录同时表明，初稿和代码更容易产生以后，组织需要投入更多能力判断哪些结果可信、哪些风险需要人工承担。

这对团队管理提出了新的要求。AI 工具的覆盖率不能单独作为成效指标；工具使用越深入，越需要明确最终责任人、跨领域评审规则和验证证据。否则，生成速度可能先于组织的校验能力增长。

7. 局限性

本研究存在以下限制：

1. 研究来自单一团队，没有同期对照组，无法排除业务周期、项目安排、人员变化和其他治理措施的共同影响。
2. 当前只有集中推进前后两个阶段的汇总日均值，缺少按周、按月和按仓库的分布数据。
3. 合并 MR 反映研发活动，不直接反映需求完成、人员投入、质量或业务价值。
4. 需求责任变化缺少可复核的分子、分母和统一判定规则，因此只作定性描述。
5. 智能评审和自动化验证主要反映能力采用，缺少缺陷、返工、回滚和生产事故等同口径结果数据。

8. 结论

在本次 181 天观察期内，团队集中推进 AI 辅助编程后，31 个业务仓库的日均合并 MR 数由 36.7 个增至 66.9 个。团队的需求责任边界和协作方式也在变化，并开始通过失败案例复盘、任务技能与提示词、多层审查、跨技术栈评审和自动化验证管理 AI 生成结果。

现阶段可以确认的是研发活动和治理方式发生了变化。是否已经提高人均生产率、软件质量和业务价值，还需要更长周期及更完整的数据。下一阶段应继续记录合并 MR 的时间序列、需求交付周期、审查投入、缺陷、返工、回滚和业务结果，检验现有机制是否真正有效。

附录：后续数据计划

数据项	最小统计口径	用途
合并 MR 时间序列	按周、按仓库统计；明确机器人和非业务变更规则	判断增长的持续性与分布
需求交付周期	从进入开发到完成交付的中位时长	判断交付是否提速
审查投入	首次响应时间、审查时长和返工轮次	判断验证成本如何变化
质量结果	缺陷、紧急修复、回滚和生产事故	判断质量是否改善
需求责任方式	需求总数、单人闭环数及统一判定规则	量化责任边界变化
业务结果	按期交付率或与需求对应的业务指标	判断研发变化是否转化为业务价值