人工智能技术对传统软件开发范式的影响研究

张 超

南京交通职业技术学院 江苏 南京 211188

【摘 要】：生成式人工智能正逐步融入软件研发工作，促使传统软件开发范式发生适应性调整。本文以人机协同开发现象作为研究对象，结合工程项目开发实践，围绕任务分工、团队协作、质量评判开展质性分析。研究发现，大模型输出代码常出现业务边界覆盖不足、项目适配程度有限等问题，工具使用不当还会逐步削弱开发者的业务推演能力。针对工程实践中的这类痛点，提出分层校验、规范工具应用、明确人机职责等可行思路，可为研发团队管控智能编码风险提供现实参考，也为人机协同下软件工程范式优化提供借鉴。

【关键词】：软件开发；人工智能；大语言模型；软件工程；开发范式

DOI:10.12417/2811-0536.26.10.095

引言

大语言模型技术落地之后，智能编码工具在软件开发领域得到普遍应用，切实提升基础编码环节效率的同时，也给项目质量管控带来新的考验。现有成果大多围绕技术能力展开讨论，较少站在工程落地角度分析工具给项目实践带来的实际变化。不少研发团队直接引入 AI 生成代码，却没有同步调整内部开发与评审流程，各类隐性问题随之产生。软件开发范式代表项目长期沉淀形成的工程实践模式，并不等同于工具技术本身。本文立足工程应用场景，结合业务开发过程中的真实现象梳理人机协同模式下出现的现实矛盾，寻找适配项目现状的处理方式，希望为一线开发团队处理这类新问题提供分析思路。

1 人工智能技术驱动软件开发范式变革的主要表现

1.1 研发任务结构发生重构

传统软件开发流程中，基础编码工作占据开发者大部分日常工时，整体工作模式具备较强的重复性和固定性。常规业务开发过程中，工作人员需要反复完成数据表字段映射、实体类定义、基础参数校验、常规数据库交互逻辑等标准化内容。这类工作技术门槛偏低，不需要复杂的业务思考，但整体耗时占比偏高，长期占用开发者用于架构设计、业务优化、难点问题攻坚的核心时间。随着软件项目迭代速度不断加快，业务需求持续更新，大量模板化编码工作容易造成人力浪费，也会限制研发人员技术能力的提升空间[1]。AI 工具普及应用后，这类标准化、重复性的编码工作可以通过模型快速完成输出，有效替代人工机械性的工作内容。开发者不用持续投入精力处理基础代码编写，工作重心慢慢向需求解读、业务逻辑设计、风险识别预判、代码优化迭代等高价值工作方向偏移。任务分工出现变化的背后，整套研发人力的价值侧重点也随之改变，行业原有固化的研发任务配比模式得到逐步调整。

1.2 团队协作链条出现深层调整

传统软件研发模式下，所有代码内容均由开发人员手动编写完成，每一段代码的逻辑设计、功能实现都经过人工完整思考，代码产出来源清晰可控。团队协作过程中，产品人员对接需求，开发人员落地功能，测试人员核验效果，整条工作链路权责清晰，问题溯源和责任划分都具备明确依据，评审工作重点也集中放在逻辑正确性、业务匹配度和代码规范性等核心维度。AI 辅助开发模式普及后，代码来源变得更加多元，工具生成的基础代码片段大量混入项目源码之中。模型产出内容仅能实现通用功能，很难贴合项目专属的业务流转规则，开发者需要对机器代码进行修改、整合和优化后再提交入库[2]。这就导致最终项目代码同时包含机器初稿和人工改写内容，评审人员很难直接分辨不同代码片段的产出主体。原有评审工作模式已经不足以覆盖这类新的开发场景，评审流程除核对基础业务逻辑，还需要额外检查模型输出部分是否适配项目整体技术体系，排查代码潜藏的隐性漏洞。代码责任归属、评审关注重点发生明显改变，团队内部长期沿用的协作模式需要随之做出适应性调整。

1.3 代码质量评判维度得到扩充

传统人工编码模式下，行业评判代码质量的标准相对固定，主要围绕程序运行稳定性、功能执行效率、

作者简介：张超，女（1979.04-），汉族，河南省南阳市，副教授，本科，研究方向为计算机及应用。

业务逻辑匹配程度、代码规范程度等核心维度展开。由于所有代码均由开发者自主设计编写，工作人员对整体业务流程和边界场景具备完整认知，代码问题大多集中在书写疏漏、经验不足等基础层面，常规测试用例基本可以覆盖绝大多数风险点。AI 编码工具介入研发流程后，原有单一的质量评价标准已经无法适配全新的代码产出形式。模型生成的代码普遍存在表层逻辑通顺、基础功能可用，但适配性不足的问题，很多代码片段无法贴合项目长期积累的专属业务约束和迭代规则。以用户积分变动业务为例，工具产出代码可以顺利完成积分增减的基础功能，却容易忽略账户冻结、重复操作、权限限制等特殊业务场景，这类隐性问题很难通过常规功能测试及时发现。项目开展质量检查工作时，工作人员不能只验证基础业务功能是否正常运行，还需要重点核对 AI 生成代码与项目整体架构、业务规则的适配程度，新增针对性的专项核查环节，进一步完善整套代码质量评判体系。

2 AI 辅助开发模式下工程实践暴露的现实矛盾

2.1 模型输出存在业务边界的疏漏，抬高后期运维负担

大语言模型依托海量公开开源代码样本完成训练，整体输出逻辑贴合通用软件开发规范，能够满足绝大多数普通业务场景的开发需求。但不同企业、不同项目会根据自身业务发展，形成独有的业务规则、数据处理方式和历史迭代逻辑，这类个性化约束并未收录在模型训练数据之中。这就导致模型无法精准掌握单一项目的小众业务场景和特殊限制条件，代码输出内容普遍存在场景覆盖不全的问题[3]。订单列表查询、用户资金退款这类核心业务，长期迭代过程中积累大量特殊处理规则，工具生成的代码仅能实现通用基础功能，极易遗漏各类小众边界条件。这类代码缺陷具备极强的隐蔽性，能够顺利通过编译阶段和常规测试阶段，不会在开发初期暴露问题。项目上线运行后，随着业务数据不断累积、特殊场景持续触发，隐性漏洞会逐步转化为线上故障。工作人员排查这类问题时，需要区分人工编码内容和机器生成内容，重新梳理完整业务链路，大幅增加故障定位和修复的人力成本，直接抬高项目后期的整体运维负担。

2.2 多来源代码随意拼接，加剧项目代码碎片化

日常开发工作中，多数开发者使用 AI 工具时，会直接选用模型输出的可用代码片段，简单调整局部内容后就并入现有项目工程，很少对整体代码风格、逻辑结构进行统一优化。不同时间段、不同开发者调用工具生成的代码，会呈现出不一样的编码风格，异常捕获方式、参数处理逻辑、变量命名习惯都存在明显差异。企业项目经过长期迭代，已经形成一套统一、稳定的编码规范和代码结构体系，新旧不同风格的代码片段混杂在同一业务文件中，会严重破坏项目代码的整体性和统一性[4]。以用户账户退款模块开发为例，模型生成的基础流程、开发者修改的核心逻辑、项目原有历史代码拼接在一起，整体功能可以正常运行，但内部代码结构杂乱无序。随着项目持续迭代开发，这类碎片化问题会不断累积，项目代码可读性持续下降，后续接手维护的工作人员需要花费大量时间梳理杂乱的逻辑，代码迭代优化和故障排查的出错概率也会明显提升，不利于项目长期稳定维护。

2.3 工具过度介入编码过程，弱化开发者业务推演意识

基础模板代码的手动编写过程，是开发者熟悉业务规则、梳理场景分支、巩固开发逻辑的重要过程，能够帮助工作人员全方位掌握业务的各类边界条件和异常场景。AI 工具替代人工完成大量基础编码工作后，部分开发者逐渐简化前期业务梳理流程，拿到开发需求后直接依靠模型生成完整代码，仅对输出内容做小幅调整就完成交付。这种开发模式跳过了自主拆解需求、推演业务逻辑、梳理异常分支的核心思考环节，开发者对整体业务流程的掌握程度会持续降低。在退款、订单结算这类逻辑复杂的业务开发中，模型容易遗漏特殊场景处理逻辑，而开发者由于缺少完整推演过程，很难主动发现代码存在的隐性缺陷。长期维持这类被动的开发模式，开发者自主分析业务、设计实现方案、排查复杂故障的能力会逐步弱化，独立解决开发难题的水平不断下降，不仅影响个人技术成长，也会制约团队整体技术能力的积累和提升。

3 面向工程场景的优化实践路径

3.1 建立分层校验机制，拦截模型带来的隐性风险

研发团队无需全盘否定 AI 工具的使用价值，可根据业务模块的影响范围和风险等级，制定差异化的工具使用和代码校验标准，以此规避模型输出带来的隐性风险。对于常规工具方法、数据格式转换、简单参数校验等低风险开发场景，模块逻辑简单、业务影响范围小，工具输出内容的安全性和适配性相对可控，开发者可以合理参考模型产出内容开展开发工作。代码提交入库前，工作人员需要主动补充各类边界场景测试，模拟异常输入、空数据、极值数据等特殊情况，验证程序在复杂场景下的运行状态，保证基础功能稳定可靠。对于订单批量查询、资金退款、账户状态流转等关联核心业务、影响用户数据安全的高风险模块，

需要严格把控工具使用尺度，不直接采纳模型生成的完整业务代码。模型输出内容仅作为开发思路参考，核心业务逻辑、数据校验规则、异常处理流程全部由开发者独立梳理和编写。代码评审阶段增设 AI 代码专项核查环节，重点检查数据查询范围、异常分支覆盖情况和业务约束落实效果，尽可能在上线前拦截各类隐性隐患，降低线上故障发生概率。

3.2 规范工具使用流程，缓解代码碎片化问题

为改善多源代码混杂带来的代码碎片化问题，团队需要建立统一的 AI 工具使用规范，明确模型代码的引入和处理标准，杜绝随意粘贴、简单修改的粗放式开发行为。开发者调用工具生成代码后，禁止直接大段拼接并入项目工程，必须逐行阅读解析代码逻辑，结合项目既有编码规范、技术架构和业务规则完成整体改写，保证新增代码与原有代码风格统一、逻辑连贯。针对用户账户变更、订单流转、资金处理等高频核心业务场景，团队可统一制定代码编写标准，规范异常捕获、事务处理、参数校验的实现方式，为开发者使用 AI 工具提供统一参考。复杂业务接口开发过程中，尽量拆分开发步骤，分段获取模型输出内容，逐段适配优化，避免一次性引入大量风格不符的机器代码[5]。代码评审环节中，评审人员重点核查 AI 拼接代码的适配性和连贯性，及时整改风格杂乱、逻辑突兀的代码片段，通过标准化流程约束，持续优化项目代码结构，保障项目长期可维护性。

3.3 明确人机工作侧重，保留开发者核心推演能力

研发团队需要清晰划分人机协同的工作边界，明确 AI 工具和开发者的工作定位，避免工具过度介入核心开发环节。模板代码编写、数据格式处理、基础语法实现等重复性工作，可依托 AI 工具完成，有效提升基础开发效率。业务需求拆解、场景分支推演、技术方案设计、核心逻辑优化等关键工作，必须由开发者独立完成，不能依托工具替代人工思考。面对退款、账户状态调整等逻辑复杂、风险较高的业务场景，开发者需要在编码工作开展前，自主梳理完整业务流转流程，梳理各类异常场景和处理方案，形成清晰的开发思路后，再借助工具完成基础编码工作。团队定期开展代码复盘和技术交流工作，针对 AI 参与开发产生的线上故障、代码漏洞开展集中分析，总结工具使用的常见问题和规避方法，帮助开发者认清模型的固有局限，树立理性的工具使用观念，在发挥工具提效优势的同时，持续锻炼和保留自身的业务推演与问题排查能力，实现开发效率与团队技术水平的同步提升。

4 结论

智能编码工具并未颠覆现有的软件工程体系，更多推动开发范式完成局部适应性演变。人机协同模式下暴露的各类工程问题，并非全部由工具本身引发，很大程度上源于项目流程未能适配新的开发模式。做好模型输出分层校验、统一工具使用规范、合理划分人机工作侧重，能够在一定程度上降低智能编码带来的质量风险。

参考文献：

[1] 戴俊峰,周搏.软件 3.0 范式变革及产业发展研判[J].中国信息化,2025,(8):46-47.

[2] 梁文仪.代码生成大语言模型的正确性与鲁棒性系统评测[J].现代信息科技,2026,10(1):68-73.

[3] 唐欢.人工智能技术在软件自动化测试中的应用[J].智慧中国,2025,(11):106-107.

[4] 李梦瑶,任萌.“人机协同”的新型教育评价模式实践探索——以软件工程专业导论为例[J].电脑知识与技术,2025,21(26):130-133.

[5] 窦浩.人工智能驱动下软件开发范式的认知革命与重构路径[J].佳木斯职业学院学报,2026,42(2):230-232.