软件过程模型	核心思想	优点	核心阶段 / 实践	缺点	适用场景
1. 瀑布模型（Waterfall）	以 “阶段线性推进” 为核心，强调文档驱动和阶段评审，前一阶段完成后才能进入下一阶段。	1. **流程清晰可控**：阶段划分明确，便于管理和进度跟踪，适合新手团队快速上手；2. **文档完整**：各阶段产出规范文档，便于后期维护、审计和知识传承；3. **早期风险暴露**：通过阶段评审提前发现需求或设计问题，降低后期返工成本；4. **责任明确**：每个阶段有明确的交付物和责任人，适合对流程合规性要求高的场景。	需求分析→可行性研究→概要设计→详细设计→编码→测试→部署→维护（每个阶段输出明确文档，如需求规格说明书、设计文档、测试报告等，需通过评审后进入下一阶段）。	1. **需求适应性差**：一旦需求变更，需回溯至对应阶段修改，后期变更成本极高（据统计，后期修改成本可能是前期的 10-100 倍）；2. **用户反馈滞后**：客户需等到项目末期才能看到可运行产品，可能存在需求误解却无法及时修正；3. **灵活性不足**：线性流程无法应对不确定性，若某一阶段延误，会直接影响整体进度；4. **过度依赖前期分析**：若需求分析不充分，后续阶段会持续受影响。	需求明确且稳定的项目（如政府信息化系统、硬件驱动开发）；对文档和合规性要求高的项目（如金融监管系统、医疗设备软件）；短期、低复杂度项目。
2. 原型模型（Prototype）	以 “快速验证需求” 为核心，通过构建可交互原型获取用户反馈，逐步迭代至最终产品。	1. **需求澄清高效**：通过可视化原型解决需求模糊问题，减少 “开发者理解与用户预期不一致” 的风险；2. **用户参与度高**：用户直接试用原型，反馈更具体，增强产品适用性；3. **降低开发风险**：提前暴露技术或设计难点（如界面交互逻辑），避免后期大规模调整；4. **灵活性较强**：原型迭代周期短（通常几天到几周），可快速响应反馈。	需求调研→快速原型设计→用户试用与反馈→原型迭代优化→最终产品开发→测试→部署（原型可分为 “抛弃式” 和 “演化式”：前者仅用于验证需求，后者可逐步演变为成品）。	1. **原型与成品混淆**：用户可能将原型误认为最终产品，对质量或功能产生过高预期；2. **开发标准可能降低**：为快速交付原型，可能忽略代码规范或架构设计，导致后期维护困难；3. **迭代成本累积**：频繁修改原型可能延长开发周期，若用户反馈不一致，会导致方向摇摆；4. **文档易缺失**：聚焦原型迭代，可能忽视需求文档整理，影响长期维护。	需求模糊或创新性强的项目（如新产品研发、用户界面设计）；用户对功能预期不明确的场景（如新型社交 APP、企业定制化工具）；技术探索性项目（如验证某一核心算法的可行性）。
3. 增量模型（Incremental）	以 “分阶段交付完整功能” 为核心，将系统拆分为多个增量，每个增量独立开发并交付，最终叠加为完整系统。	1. **早期交付价值**：首个增量即可交付核心功能，让用户提前受益（如电商平台先交付 “商品浏览 + 下单”，再迭代 “支付 + 物流”）；2. **风险分散**：每个增量独立测试和交付，避免 “一次性开发失败” 的风险；3. **资源可控**：可根据市场反馈调整后续增量优先级，优化资源分配；4. **便于集成验证**：增量间接口提前设计，降低后期系统集成难度。	整体需求分析→架构设计→增量 1（需求→设计→开发→测试→交付）→增量 2（同上）→...→增量 N→系统集成→维护（每个增量需包含可运行的端到端功能，且依赖前期架构规划）。	1. **架构设计要求高**：需提前规划整体架构和增量间接口，若设计不合理，会导致增量难以兼容（如数据格式冲突）；2. **增量依赖强**：后续增量可能依赖前期增量的功能，若前期增量修改，会引发连锁反应；3. **用户反馈周期较长**：增量周期通常为几周到数月，对快速变化的需求响应较慢；4. **团队协作复杂**：多增量并行开发时，需协调进度和资源，避免冲突。	大型复杂项目（如企业 ERP 系统、银行核心系统）；需分阶段交付的项目（如按季度迭代的 SaaS 产品）；资源有限但需尽早见成效的场景（如创业公司的核心功能优先上线）。
4. XP 模型（极限编程）	以 “拥抱变化、持续反馈” 为核心，通过简化流程和强化团队协作，应对高频需求变更，聚焦代码质量和快速交付。	1. **需求适应性极强**：通过短周期迭代和客户实时参与，可快速响应需求变化（如市场突发调整）；2. **代码质量高**：结对编程和 TDD 大幅减少 Bug（据统计，TDD 可降低 30%-50% 的缺陷率），重构避免技术债务；3. **团队协作高效**：面对面沟通和集体代码所有权促进知识共享，减少信息壁垒；4. **用户价值优先**：每个发布版本聚焦核心功能，确保交付内容符	核心实践包括：・结对编程（两人协作编码，实时审查）；・TDD（测试驱动开发，先写测试用例再编码）；・持续集成（每日多次合并代码并自动化测试）；・小型发布（2-4 周交付可运行版本）；・重构（优化代码结构）；・客户现场参与（实时解答需	1. **对团队要求严苛**：依赖成员的技术能力和协作意识（如结对编程初期效率可能下降 50%），新手团队难以适应；2. **文档简化的隐患**：侧重口头沟通，文档较少，长期维护或人员变动时可能出现知识断层；3. **客户参与成本高**：要求客户全程投入，若客户无法配合，反馈机制会失效；4. **不适合大型项目**：缺乏结构化架构规划，多团队协作时易出现混乱。	需求频繁变化的中小型项目（如互联网创业产品、移动 APP）；团队规模小（5-15 人）且技术能力强的场景；代码质量要求高的领域（如金融交易系统、医疗设

		合用户实际需求。	求）。		备控制软件）。
5. Scrum 模型	以 “固定迭代周期（Sprint）” 为核心，通过结构化仪式（如站会、评审会）实现团队自组织，聚焦可交付成果和持续改进。	1. **迭代节奏明确**：固定周期确保团队专注于短期目标，避免进度拖延；2. **透明度高**：通过待办列表和每日站会，项目进度对团队和客户可见，减少信息不对称；3. **团队自组织性强**：成员自主分配任务，责任感和创造力更强；4. **持续改进机制**：回顾会定期优化流程，提升团队效率（如减少会议时间、改进沟通方式）。	核心流程：・产品待办列表（Product Backlog，梳理需求优先级）；・Sprint 规划会（确定迭代目标和任务）；・每日站会（15 分钟同步进度、问题和计划）；・Sprint 评审会（向客户演示成果并收集反馈）；・Sprint 回顾会（总结经验并优化流程）；迭代周期通常为 2-4 周。	1. **对 Product Owner 依赖高**：需有明确的需求负责人（Product Owner），若其对业务理解不足，会导致迭代目标偏离；2. **仪式执行不当易形式化**：如每日站会超时或流于表面，反而降低效率；3. **文档简略**：侧重交付成果，设计文档可能不足，长期维护需依赖团队经验；4. **大型项目协调难**：多团队并行 Scrum 时，需额外机制（如 Scrum of Scrums）协调，复杂度高。	需求动态变化的项目（如电商平台迭代、企业 SaaS 产品更新）；需要快速验证市场的场景（如初创公司的 MVP 开发）；团队成熟度中等以上，能自主管理的项目。

传统模型与敏捷模型核心差异总结：

传统模型（如瀑布、原型、增量）以 “预测性” 为核心，强调通过前期详尽规划锁定需求与流程，依赖完整文档驱动开发，各阶段界限分明（如瀑布的线性阶段、增量的预设模块划分）。其优势在于流程可控、文档规范，适合需求稳定、合规性要求高的场景（如政府项目），但对后期需求变更的适应性差，修改成本随项目推进呈指数级增长。

敏捷模型（如 XP、Scrum）以 “适应性” 为核心，弱化前期规划，通过短周期迭代（如 Scrum 的 Sprint、XP 的小型发布）快速交付可用成果，依赖团队高频协作（如每日站会、结对编程）和用户实时反馈调整方向。其优势在于能灵活响应市场变化，适合需求动态的互联网产品，但需团队具备强自组织能力，且因文档简化可能给长期维护带来挑战。