2352835 夏弘泰

各模型分析表格

软件过程模型	核心思想	优点	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；2. 文档完整；3. 早期风险暴露；4. 责任明确。	1. 需求适应性差；2. 用户反馈滞后；3. 灵活性不足；4. 过度依赖前期分析。	需求明确、流程合规要求高（如政府、金融、医疗系统）。
2. 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。	1. 早期交付价值；2. 风险分散；3. 资源可控；4. 便于集成验证。	1. 架构设计要求高；2. 增量依赖强；3. 用户反馈周期长；4. 团队协作复杂。	大型复杂项目、分阶段交付项目、资源有限但需早期上线场景。
3. 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。	1. 需求澄清高效；2. 用户参与度高；3. 降低开发风险；4. 灵活性强。	1. 原型与成品混淆；2. 开发标准降低；3. 迭代成本累积；4. 文档易缺失。	需求模糊或创新性强的项目（如新产品研发、界面设计）。
4. 螺旋模型（Spiral）	以“风险驱动的迭代”为核心，将瀑布的阶段化流程与原型的迭代反馈结合，通过多轮螺旋循环不断细化系统。	1. 风险控制强：在每轮迭代前分析并消除风险，降低失败概率；2. 灵活性高：结合瀑布的系统性与原型的迭代性；3. 用户参与持续：每轮评审保障用户反馈融入设计；4. 适应性强：可根据风险和需求调整迭代内容。	1. 成本高：频繁风险分析与评审增加管理负担；2. 实现复杂：对项目管理与技术能力要求极高；3. 不适合小项目：前期风险评估成本难以摊销；4. 文档量大：每轮均需详细记录。	高风险、大规模、技术复杂的项目（如军工、航天、操作系统、银行系统）。
5. 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。	1. 需求适应性强；2. 代码质量高；3. 协作高效；4. 用户价值优先。	1. 对团队要求高；2. 文档简化隐患；3. 客户参与成本高；4. 不适合大型项目。	需求频繁变化的中小型项目（如互联网产品、APP 开发）。

软件过程模型	核心思想	优点	缺点	适用场景
6. Scrum 模型	以 “固定迭代周期（Sprint）” 为核心，通过结构化仪式实现团队自组织。	1. 迭代节奏明确；2. 透明度高；3. 自组织性强；4. 持续改进机制。	1. 对 PO 依赖高；2. 仪式易形式化；3. 文档简略；4. 大型项目协调难。	需求动态变化、需快速验证市场的项目（如 SaaS、MVP 开发）。
7. 看板模型（Kanban）	以 “可视化工作流与持续交付” 为核心，通过限制在制品数量（WIP）与任务流动控制，持续优化开发效率。	1. 流程可视化强：一眼可见任务状态与瓶颈；2. 持续交付：无需等待迭代周期，可随时发布；3. 高适应性：适合需求不断变化场景；4. 团队负载均衡：WIP 限制防止过载。	1. 目标模糊风险：缺少固定迭代可能导致缺乏短期目标；2. 过度自由：需自律团队，否则效率波动大；3. 难以追踪长期规划：对产品路线图支持弱；4. 依赖文化氛围：若缺乏持续改进意识，效果不显著。	运维、技术支持、持续交付项目（如 DevOps 团队、服务型团队）；需求频繁变动、任务碎片化的场景（如线上运维、Bug 修复）。

传统模型与敏捷模型分析

传统模型（如瀑布、增量等）

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（如增量的预设模块划分等）。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。

原型模型 (介于传统模型与敏捷模型之间的过渡模型)

以 “需求验证” 为核心，通过快速构建可交互原型获取用户反馈，迭代优化至需求清晰，既不依赖严格的前期规划，也未形成系统化的持续交付框架。其优势在于解决需求模糊问题，用户参与度高，降低初期需求误解风险，但迭代多为局部优化，易因原型与成品混淆导致预期偏差，适合嵌入传统或敏捷模型中辅助需求澄清。

敏捷模型（如 XP、Scrum等）

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。
