工作量估计

项目组工作风格：外向（告诉他人你的想法）+理性（决策基于事实和可能的情况谨慎考虑）

**COCOMO模型（构造性成本模型）**

pm=A×(KLOC)^B×Ⅱ(EM)

PM为工作量，通常表示为人月；

A为校准因子；

KLOC源代码程序长度的测量；

B为对工作量呈指数或非线性影响的比例因子；

EM为影响软件开发工作量的其它因素，也称调节因子

模型等级选择：

基本模型 1.在项目相关信息极少的情况下使用。2.静态单变量模型

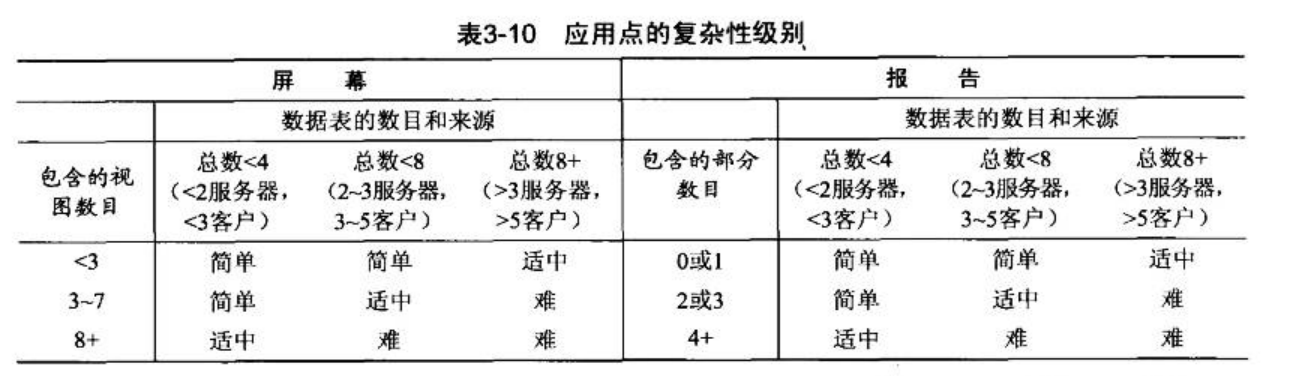
项目类型：有机1.各类应用程序，例如数据处理、科学计算等。2.受硬件的约束比较小，程序的规模不是很大。

**静态单变量模型**

E=A+B×(ev)^C ev是估算变量（KLOC或FP）

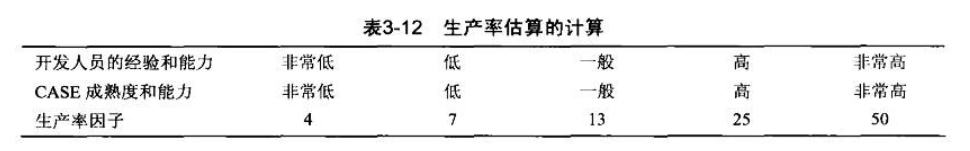
**工作量**

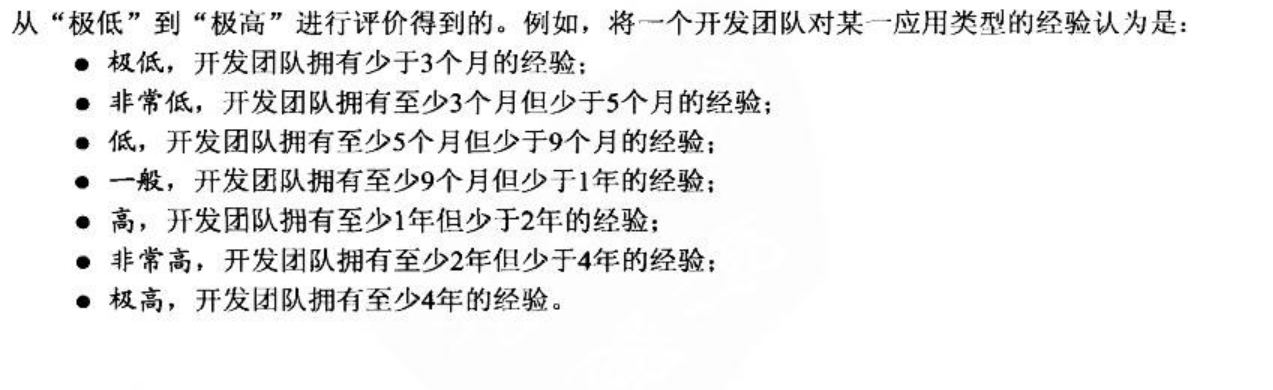
**E=3.2×(KLOC)^1.05=17.34人月**

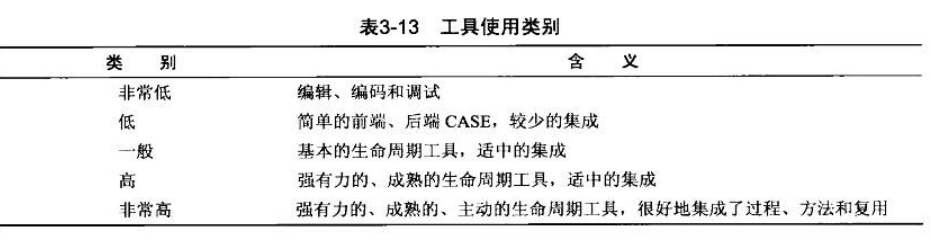
****

|  |  |  |  |
| --- | --- | --- | --- |
| 应用点 | 类别 | 程度 | 复杂性权重 |
| 用户注册与登录 | 屏幕 | 适中 | 2 |
| 书籍浏览与选购 | 屏幕 | 难 | 3 |
| 订单管理 | 报告 | 难 | 8 |
| 库存管理 | 报告 | 适中 | 5 |
| 客户留言反馈 | 屏幕 | 简单 | 1 |
| 财务管理 | 报告 | 难 | 8 |
| 网站信息维护 | 报告 | 适中 | 5 |
| 开通会员 | 屏幕 | 简单 | 1 |
| 客户数据维护 | 报告 | 难 | 8 |
| 第三代语言构件 | JAVA |  | 10 |

应用点技术NOPS=51







由于开发团队之前有过数据库等项目经历，经验认为是非常低，同时开发项目使用工具基础功能支持编码、调试等，认为是工具使用类别是非常低，从而生产率估计为4

COCOMOII模型估计时间为：51/4=12.75人月

风险管理

## **7.4技术风险评价与应对**

进行技术风险评价与应对的意义在于提前识别和解决项目中可能存在的技术问题（如系统漏洞、性能瓶颈等），通过制定针对性措施（如使用成熟技术、定期测试）降低技术失败概率，确保项目按时、高质量交付，同时减少因技术问题导致的成本超支和用户信任损失。

### **7.4.1核心风险点**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 风险类型 | 风险描述 | 可能性 | 影响程度 | 应对措施 |
| 支付集成接口 | 支付宝/微信支付API复杂度高 | 高 | 严重 | 外包开发或者使用第三方支付SDK |
| 数据安全 | 用户隐私泄露或SQL注入漏洞 | 中 | 严重 | 引入安全顾问，使用ORM框架防护 |
| 高并发处理 | 促销活动时服务器崩溃 | 低 | 中等 | 采用云服务弹性扩容，简化初期架构 |
| 多端兼容性 | 移动端适配不足 | 高 | 中等 | 优先响应式设计，牺牲部分UI复杂度 |

### **7.4.2关键补救措施**

支付集成接口：外包开发或使用第三方支付SDK（如Ping++），降低集成复杂度，确保支付功能稳定。

数据安全：引入安全顾问进行审计，采用ORM框架防止SQL注入，加密用户隐私数据，定期进行安全测试。

高并发处理：采用云服务弹性扩容功能（如AWS Auto Scaling），简化初期架构，确保促销期间系统稳定。

多端兼容性：优先采用响应式设计，确保移动端适配，适当降低UI复杂度以提升兼容性。

进行风险管理可以通过提前识别和应对潜在风险，避免或减少项目在时间、成本、质量等方面的损失。且可以为项目团队提供风险信息，帮助做出更科学、更可靠的决策。除此之外还可以将有限的资源集中在最关键的风险上，避免资源浪费。通过系统化的风险管理流程，提升对项目的掌控能力，减少意外情况的发生。并且明确的风险应对措施可以增强团队成员的信心，减少焦虑和不确定性。

## **9.1风险分类及分级**

|  |  |  |  |
| --- | --- | --- | --- |
| 风险类型 | 高风险 | 中风险 | 低风险 |
| 技术风险 | 支付系统的安全漏洞，可能会导致用户金钱损失 | 高并发场景性能不足，导致暂时的用户访问延迟过高，软件崩溃等问题 | 第三方API集成延迟，可能会影响使用体验 |
| 团队风险 | 团队成员流失，可能会有部分成员失去信心离开团队 | 任务分配不均，部分成员工作压力过大 | 沟通效率低下，团队成员之间的交流可能不够频繁，无法及时获取各部分开发进程 |
| 时间风险 | 项目进度严重延误，软件无法按时发布 | 关键功能开发延期，软件无法高效使用 | 文档编写滞后，软件开发使用相关信息无法及时保留更新 |
| 需求风险 | 用户需求理解偏差，货不对板导致用户公信力严重下降 | 功能优先级调整频繁，代码编写人员反复调整返工，加大工作量 | 用户反馈收集不足，无法进行及时有效的更新优化导致用户粘性不足 |
| 资源风险 | 开发工具成本超支，软件发布后入不敷出，团队离散，资金断流 | 服务器资源不足，导致软件在使用高峰期产生卡顿、延迟、崩溃等现象 | 学习资料获取困难，团队开发过程四处碰壁，延长了开发时间 |

## **9.2各级风险应对措施**

分级产生应对措施可以优化资源分配，提高风险管理效率。通过将风险分为高、中、低三个级别。针对各级风险的应对措施，总方针为“集中力量优先解决高风险问题，其次解决中风险问题，剩余力量解决低风险问题”。可以将有限的资源（时间、资金、人力）集中在最关键的风险上，避免低风险问题占用过多资源。针对不同级别风险制定差异化的应对策略，确保快速、精准地解决问题。避免对所有风险“一刀切”，减少不必要的管理开销。

### **9.2.1高风险应对措施**

针对上述提出的五种可能出现的高风险现象，分别采取不同的针对性措施：

（1）支付系统安全漏洞。采用成熟的第三方支付接口（如支付宝、微信支付），避免自研支付系统；定期进行安全测试（如渗透测试），确保数据传输加密（HTTPS）。设置异常交易监控机制，发现异常立即冻结账户并通知管理员。

（2）团队成员流失。明确分工并建立文档化的工作流程，确保每个成员的任务可追溯；定期进行代码审查和知识共享，避免知识孤岛。每周召开团队会议，了解成员状态，及时调整任务分配。

（3）项目进度严重延误。制定详细的项目计划，明确每个阶段的时间节点。使用甘特图（如Microsoft Project）跟踪进度，及时发现延误。预留缓冲时间，应对不可预见的延误。每周评审项目进度，调整计划以确保按时交付。

（4）用户需求理解偏差。通过用户调研和访谈，明确用户需求。使用用户故事（User Story）描述需求，确保开发团队理解一致。定期与用户沟通，验证需求实现情况。通过A/B测试验证功能是否符合用户需求。

（5）开发工具成本超支。优先使用开源工具（如Git、Jenkins）降低开发成本。制定预算计划，严格控制工具采购成本。寻找免费或低成本替代方案（如使用社区版IDE）。定期评审开发工具使用情况，确保成本可控。

### **9.2.2中风险应对措施**

针对上述提出的五种可能出现的中风险现象，同样采取不同的针对性措施：

（1）高并发场景性能不足。使用缓存技术（如Redis）优化数据库查询，减少数据库压力。采用负载均衡技术（如Nginx）分散流量，避免单点故障。提前进行压力测试，模拟高并发场景，优化系统性能。监控：部署性能监控工具（如Prometheus），实时跟踪系统响应时间。

（2）任务分配不均。使用项目管理工具（如Trello或Jira）可视化任务分配，确保工作量均衡。定期轮换任务，避免单一成员负担过重。根据成员能力合理分配任务，避免能力与任务不匹配。每日站会（Scrum）同步任务进度，及时发现并解决问题。

（3）关键功能开发延期。优先开发核心功能（如用户注册、图书检索），确保基本功能可用。使用迭代开发模式（如Scrum），分阶段交付功能。定期进行代码审查，确保开发质量。每日站会同步开发进度，及时发现并解决问题。

（4）功能优先级调整频繁。使用优先级矩阵（如MoSCoW法）明确功能优先级。定期与用户沟通，确认优先级调整的必要性。预留开发资源，应对优先级调整带来的额外工作量。通过项目管理工具跟踪功能优先级变化。

（5）服务器资源不足。使用云服务（如AWS、阿里云）动态扩展服务器资源。部署监控工具（如Zabbix），实时跟踪服务器负载预留应急资金，用于购买额外务器资源。定期检查服务器资源使用情况，及时扩容。

### **9.2.3低风险应对措施**

针对上述提出的五种可能出现的低风险现象，同样采取不同的针对性措施：

（1）第三方API集成延迟。选择稳定的第三方服务商，制定备选方案（如本地模拟数据）。在合同中明确服务响应时间，确保API可用性。定期测试API接口，确保其正常运行。设置API健康检查机制，发现异常时切换至备选方案。

（2）沟通效率低下。使用协作工具（如Slack或钉钉）提高沟通效率。定期召开团队会议（如每周一次），同步开发进度。建立文档化沟通流程，确保信息传递准确，通过项目管理工具跟踪沟通任务完成情况。

（3）文档编写滞后。在每次迭代结束后统一补充文档，确保文档与代码同步。使用自动化工具（如Swagger）生成部分文档，减少手动编写工作量。设置文档编写任务，纳入迭代计划中。定期检查文档完成情况，确保文档及时更新。

（4）用户反馈收集不足。在软件中集成用户反馈功能（如评分、评论）。定期分析用户反馈数据，识别改进点。通过社交媒体或邮件收集用户意见。监控：定期评审用户反馈，制定优化计划。

（5）学习资料获取困难。利用在线学习平台（如Coursera、慕课）获取学习资源。建立团队内部知识库，共享学习资料和经验。定期组织技术分享会，提升团队技能。定期检查团队成员学习进度，确保技能提升。

## **9.3风险管理工具与方法**

在项目管理中，风险管理是确保项目顺利推进的关键环节。通过科学的风险评估和有效的管理工具，团队可以提前识别潜在问题并制定应对策略，从而降低风险对项目的影响。可以使用风险矩阵、看板系统、定期评审以及应急资金预留等方法，构建一套全面的风险管理体系，以应对项目中的不确定性。

（1）风险矩阵。使用风险矩阵评估每个风险的发生概率和影响程度，确定优先级。例如，支付系统安全漏洞属于高概率、高影响，应优先处理。

（2）看板系统。使用Trello或Jira等工具，将风险任务可视化，实时跟踪风险状态。例如，设置“风险看板”，标注每个风险的当前状态（如“未解决”“处理中”“已解决”）。

（3）定期评审。每周召开风险评审会议，评估已识别风险的变化情况，并讨论新风险。例如，发现某个API接口响应变慢时，立即启动备选方案。

（4）应急资金与资源预留。为高风险预留应急资金（如购买服务器资源或支付安全测试服务），确保问题发生时能快速响应。

综上所述，风险管理是项目成功的重要保障。通过风险矩阵评估风险的优先级，利用看板系统实时跟踪风险状态，结合定期评审会议动态调整应对策略，以及为高风险预留应急资金和资源，团队能够系统化地应对项目中的潜在威胁。这些方法不仅提升了风险管理的效率，还为项目的稳定推进提供了有力支持。