实验四

实验目的：

1. 了解XP开发方法
2. 了解DevOps
3. 理解项目活动图

实验内容：

1. 阅读XP开发方法文档，理解XP过程工作模型

Extreme Programming（极限编程，简称XP）是由KentBeck在1996年基于增量模型发展而提出的。是一种近螺旋式的开发方法，将复杂的开发过程分解为一个个相对比较简单的小周期，将系统细分为多个可以在较短周期解决的子模块，且强调测试、代码质量和及早的发现问题。通过积极的交流、反馈以及其它一系列的方法，开发人员和客户可以非常清楚开发进度、变化、待解决的问题和潜在的困难等，并根据实际情况及时地调整开发过程。

XP过程工作模型包括四个主要方面：价值、流程、角色和实践。

价值：

价值驱动（Value-Driven）：XP注重以客户价值为导向的开发，将客户需求和业务目标置于核心位置。通过不断地与客户合作和反馈，团队能够快速响应需求变化，并提供高价值的软件解决方案。

流程：

计划游戏（Planning Game）：团队与客户一起进行规划会议，讨论需求、优先级和估算，制定迭代计划。在计划游戏中，团队与客户保持紧密的沟通和协作，以确保共同理解和共识。

小步快跑（Small Releases）：XP鼓励频繁、可交付的软件发布。通过迭代周期内的小规模交付，团队能够快速验证软件功能和收集用户反馈，减少开发过程中的风险。

程序员自主（Programmer Empowerment）：XP鼓励程序员的自主性和主动性。程序员可以自行安排工作，参与需求分析和设计，并在团队内部进行知识分享和代码审查。

持续集成（Continuous Integration）：团队成员频繁地将代码集成到共享的代码库中，并进行自动化的构建和测试。持续集成确保代码的一致性和质量，并尽早发现和解决问题。

测试驱动开发（Test-Driven Development）：XP倡导在编写代码之前先编写测试用例，并通过测试驱动开发的方式进行迭代开发。测试驱动开发可以提高代码质量、可维护性和可测试性。

角色：

客户（Customer）：代表业务和用户需求，与团队紧密合作，提供需求反馈和验证软件交付。

程序员（Programmer）：负责编写、测试和交付软件功能。程序员具有高度的技术能力和自主性，参与需求讨论和设计过程。

测试人员（Tester）：负责编写和执行测试用例，确保软件质量和功能符合预期。

整合者（Integrator）：负责持续集成和构建过程，确保代码的一致性和可部署性。

教练（Coach）：作为团队的指导者和支持者，提供教练和指导，促进团队的学习和成长。

实践：

简单设计（Simple Design）：XP强调简单和可演化的设计。通过持续重构和代码整理，确保系统的灵活性和可维护性。

集体代码所有权（Collective Code Ownership）：团队成员共同拥有代码，并对其质量和演化负责。任何人都可以修改和改进代码。

反馈循环（Feedback Loop）：XP注重及时的反馈和学习。通过客户反馈、测试结果和团队反思，不断改进软件和开发过程。

2. 阅读DevOps文档，了解DevOps

DevOps是一种软件开发和运维的方法论，旨在通过改进软件交付流程、强调协作和自动化，实现快速、可靠的软件交付。包括如下过程：

持续计划：

业务需求识别：与业务团队合作，识别关键业务需求和目标。这有助于制定开发和交付的优先级，并确保软件解决方案与业务需求相匹配。

持续需求分析：将业务需求转化为具体的软件功能和特性，并与开发团队共享。这有助于确保开发团队对需求有充分的理解，并为开发过程提供明确的指导。

持续开发：

版本控制：使用版本控制系统（如Git）管理源代码和开发过程。版本控制确保团队成员可以协同工作，跟踪代码更改，并恢复到之前的版本（如果需要）。

自动化构建：通过自动化构建工具（如Jenkins、Travis CI等），实现持续集成和自动化构建。这包括编译、静态代码分析、单元测试和打包等过程。

自动化测试：通过自动化测试框架（如Selenium、JUnit等），实现自动化测试和回归测试。这有助于快速检测和修复代码缺陷，确保软件质量和稳定性。

持续集成：将团队成员的代码集成到共享代码库中，并进行自动化构建和测试。这有助于尽早发现和解决集成问题，并减少代码集成时的冲突。

持续交付：

部署自动化：通过自动化部署工具（如Ansible、Docker、Kubernetes等），实现应用程序的自动化部署和配置。这有助于快速、可靠地将软件交付到目标环境，并减少人为错误。

环境管理：使用基础设施即代码（Infrastructure as Code）的概念，通过自动化工具（如Terraform、CloudFormation等）实现环境的自动化管理和配置。这有助于确保开发、测试和生产环境的一致性和可重复性。

持续监控：通过实时监控和日志分析工具（如Prometheus、ELK Stack等），监测应用程序和基础设施的性能、可用性和安全性。这有助于快速识别和解决潜在问题，并改进应用程序的运行质量。

持续反馈：

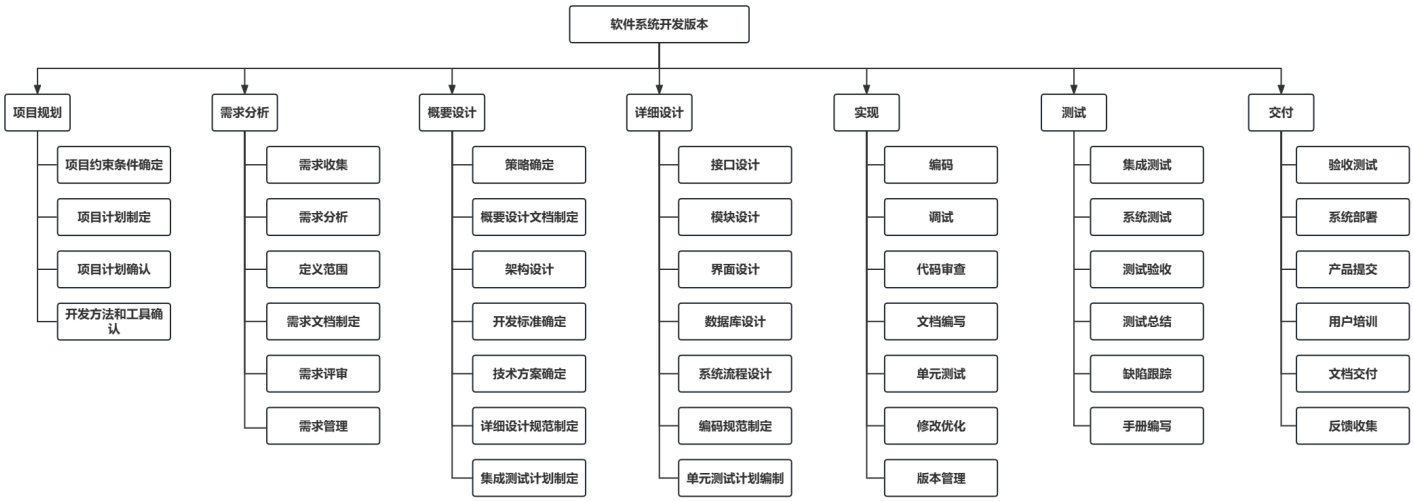
用户反馈：与用户和客户保持紧密联系，收集用户反馈和需求变更。这有助于了解用户需求，改进软件功能，并及时调整开发优先级。

运维反馈：与运维团队合作，收集应用程序的运行数据、故障报告和性能指标。这有助于快速响应和解决潜在的运行问题，并改进软件交付的可靠性和稳定性。

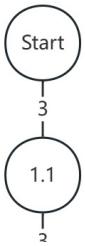
3. 活动图练习

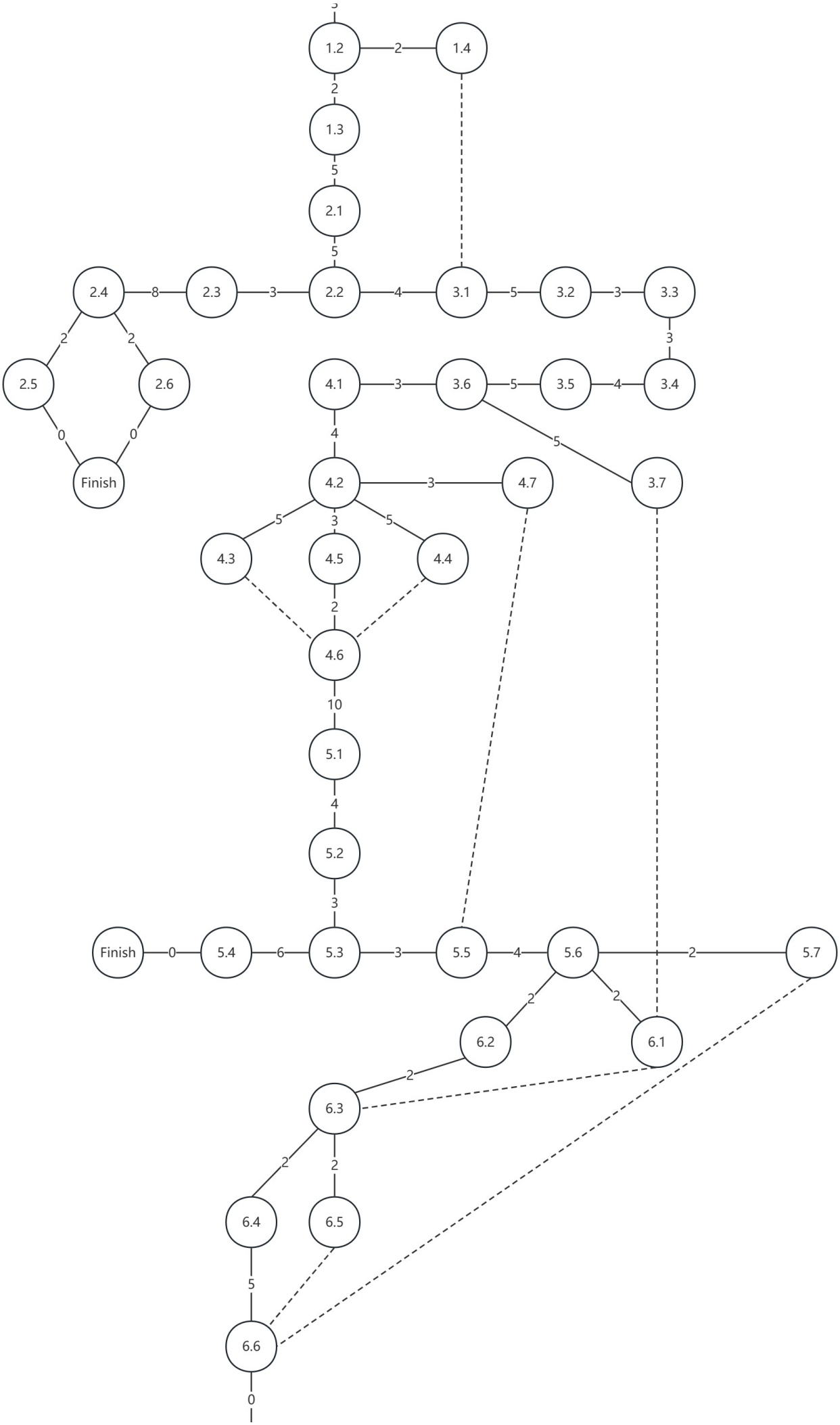
小组讨论，针对自己项目中的工作进行工作活动分解，分工进行各自合理的工作进度估算，最后汇总绘出项目活动图，找出关键路径。

WBS



工作分解结构与活动图

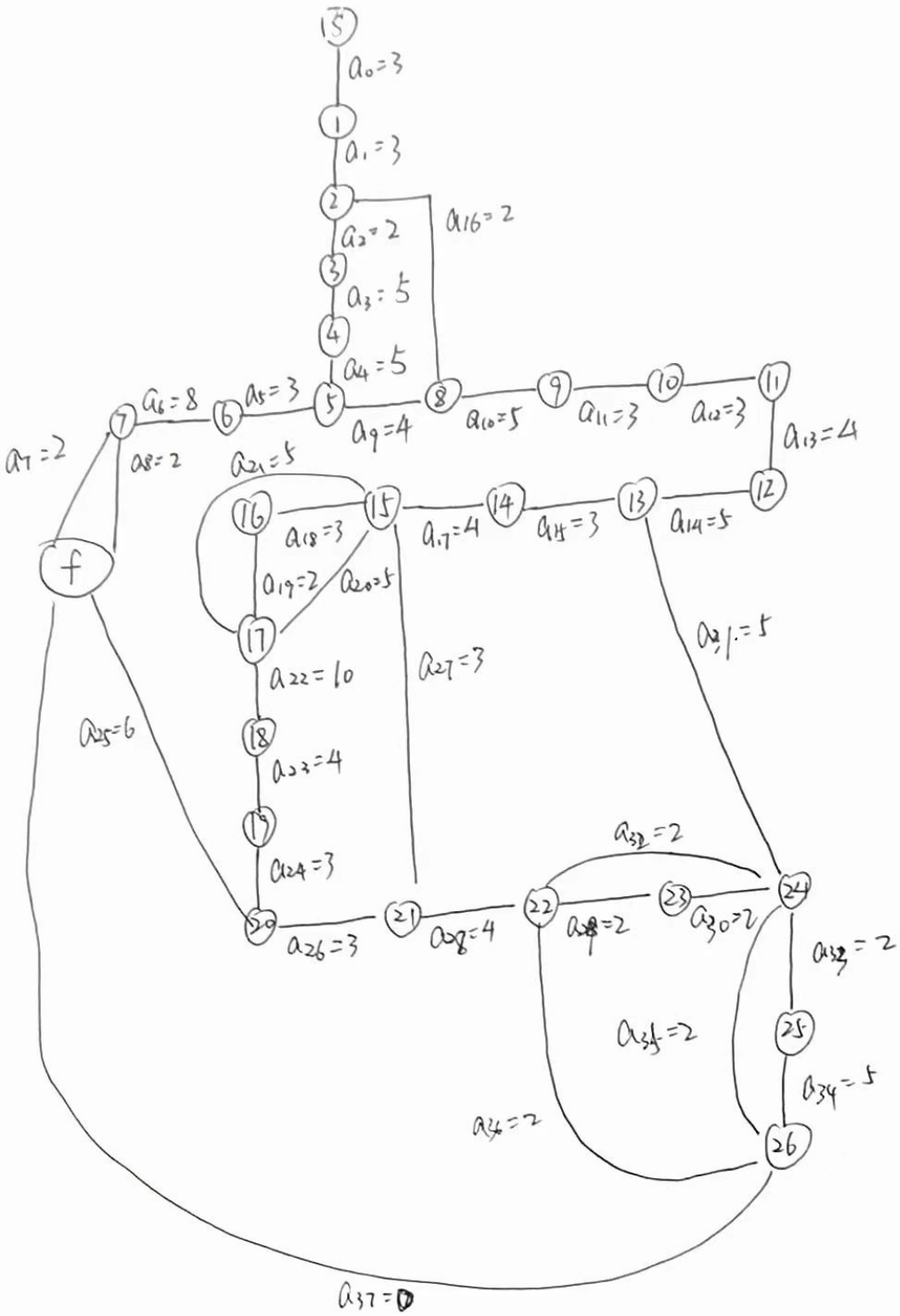



1.2
2
1.4
2
1.3
5
2.1
5
2.4
8
2.3
3
2.2
4
3.1
5
3.2
3
3.3
3
2
2
2.5
2.6
4.1
3
3.6
5
3.5
4
3.4
0
0
4
5
Finish
4.2
3
4.7
3.7
5
3
5
4.3
4.5
4.4
2
4.6
10
5.1
4
5.2
3
Finish
0
5.4
6
5.3
3
5.5
4
5.6
2
5.7
2
2
6.2
6.1
2
6.3
2
2
6.4
6.5
5
6.6
0

进展估计图

任务	预计完成时间（天）
步骤 1：项目规划	
任务 1.1：项目约束条件确定	3
任务 1.2：项目计划制定	3
任务 1.3：项目计划确认	2
任务 1.4：开发方法和工具确认	2
步骤 2：需求分析	
任务 2.1：需求收集	5
任务 2.2：需求分析	5
任务 2.3：定义范围	3
任务 2.4：需求文档制定	8
任务 2.5：需求评审	2
任务 2.6：需求管理	2
步骤 3：概要设计	
任务 3.1：策略确定	4
任务 3.2：概要设计文档制定	5
任务 3.3：架构设计	3
任务 3.4：开发标准确定	3
任务 3.5：技术方案确定	4
任务 3.6：详细设计规范制定	5
任务 3.7：集成测试计划制定	5
步骤 4：详细设计	
任务 4.1：接口设计	3
任务 4.2：模块设计	4
任务 4.3：界面设计	5
任务 4.4：数据库设计	5
任务 4.5：系统流程设计	3
任务 4.6：编码规范设计	2
任务 4.7：单元测试计划编制	3
步骤 5：实现	
任务 5.1：编码	10
任务 5.2：调试	4
任务 5.3：代码审查	3
任务 5.4：文档编写	6
任务 5.5：单元测试	3
任务 5.6：修改优化	4
任务 5.7：版本管理	2
步骤 6：测试	
任务 6.1：集成测试	2
任务 6.2：系统测试	2
任务 6.3：测试验收	2
任务 6.4：测试总结	2
任务 6.5：缺陷跟踪	2
任务 6.6：手册编写	5

关键路径



关键路径：$\langle a_0, a_1, a_2, a_3, a_4, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{17}, a_{18}, a_{19}, a_{22}, a_{23}, a_{24}, a_{26}, a_{28}, a_{29}, a_{30}, a_{33}, a_{34}, a_{37}\rangle$