**假设我们有一个简单的项目，需要开发一个学生管理系统，具有以下模块或功能点：**

学生信息管理：包括学生的基本信息、成绩和出勤记录等。

课程管理：管理课程信息和课程表。

成绩统计：对学生的成绩进行统计和分析。

教师管理：管理教师的信息和授课安排。

班级管理：管理学生所在的班级和班级的相关信息。

现在我们使用分解-累计估算法来估算这个项目的软件系统规模和编码阶段的工作量。

**划分模块或功能点：**学生信息管理、课程管理、成绩统计、教师管理、班级管理

**确定基准模块规模：**假设我们选择学生信息管理作为基准模块，估算其代码行数为200行。

**估算其他模块规模：**根据每个模块相对于基准模块的复杂程度，进行逐个模块的规模估算。假设课程管理和成绩统计模块相对于基准模块的复杂程度分别为0.8和0.6，教师管理和班级管理模块相对于基准模块的复杂程度分别为0.7和0.5。

学生信息管理：200行

课程管理：200 \* 0.8 = 160行

成绩统计：200 \* 0.6 = 120行

教师管理：200 \* 0.7 = 140行

班级管理：200 \* 0.5 = 100行

累计规模：将每个模块的规模累加起来，得到整个软件系统的总体规模。

总体规模 = 200 + 160 + 120 + 140 + 100 = 720行

计算编码阶段工作量：根据编程人员的平均工作效率（300行代码/天），将总体规模转换为工作量（人天）。  **工作量 = 总体规模 / 编码人员的平均工作效率** = 720 / 300 ≈ 2.4人天

这是一个简单的例题，实际的估算过程可能更加复杂，需要考虑更多的因素。但是通过分解-累计估算法，我们可以得到一个粗略的软件系统规模估计和编码阶段的工作量估计，以供项目规划和资源分配时参考。

**据以往项目的经验以及题目2中软件系统的规模估算，估计本项目中所有活动的持续时间及开始和结束时间？**

估计项目中所有活动的持续时间及开始和结束时间需要考虑多个因素，包括项目的复杂性、团队规模、资源分配、技术难度等。没有具体的项目细节和参数，下面是一个简单的示例来估计本项目中活动的持续时间及开始和结束时间：

假设在本项目中，我们有以下活动（以及估计的持续时间）：

需求分析和规划：2周

设计和架构：3周

编码和单元测试：4周

集成和系统测试：2周

用户验收测试：1周

修复和优化：1周

文档编写和培训：1周

为了确定活动的开始和结束时间，我们需要考虑活动之间的依赖关系和资源可用性。

假设我们从某个特定的日期作为项目开始日期，那么可以按照以下顺序安排活动的开始和结束时间：

需求分析和规划：从项目开始日期开始，持续2周。

开始时间：项目开始日期

结束时间：项目开始日期 + 2周

设计和架构：在需求分析和规划完成后开始，持续3周。

开始时间：需求分析和规划结束时间

结束时间：需求分析和规划结束时间 + 3周

编码和单元测试：在设计和架构完成后开始，持续4周。

开始时间：设计和架构结束时间

结束时间：设计和架构结束时间 + 4周

集成和系统测试：在编码和单元测试完成后开始，持续2周。

开始时间：编码和单元测试结束时间

结束时间：编码和单元测试结束时间 + 2周

用户验收测试：在集成和系统测试完成后开始，持续1周。

开始时间：集成和系统测试结束时间

结束时间：集成和系统测试结束时间 + 1周

修复和优化：在用户验收测试完成后开始，持续1周。

开始时间：用户验收测试结束时间

结束时间：用户验收测试结束时间 + 1周

文档编写和培训：在修复和优化完成后开始，持续1周。

开始时间：修复和优化结束时间

结束时间：修复和优化结束时间 + 1周

**下面是一个示例项目的工作分解结构（WBS）的列表形式：**

**项目管理** 1.1 项目规划 1.2 项目进度管理 1.3 项目风险管理 1.4 项目沟通管理

**需求分析和规划** 2.1 收集和分析用户需求 2.2 编写需求文档 2.3 确定项目范围和目标 2.4 制定项目计划和时间表

**设计和架构** 3.1 确定系统架构和技术方案 3.2 设计数据库结构 3.3 设计用户界面 3.4 编写详细设计文档

**编码和单元测试** 4.1 开发学生信息管理模块 4.2 开发课程管理模块 4.3 开发成绩统计模块 4.4 开发教师管理模块 4.5 开发班级管理模块 4.6 进行单元测试和调试

**集成和系统测试** 5.1 将各个模块集成到一个系统中 5.2 进行系统测试和验证 5.3 修复和调优系统问题

**用户验收测试** 6.1 邀请用户参与测试 6.2 收集用户反馈 6.3 修复和优化系统问题

**项目文档和培训** 7.1 编写用户手册和技术文档 7.2 进行培训和知识转移

**一级质量特性：**

功能性（Functionality）：

适用性（Suitability）：软件是否满足用户需求和预期的功能。

准确性（Accuracy）：软件的功能是否能够产生正确的结果。

互操作性（Interoperability）：软件与其他系统的交互能力。

可靠性（Reliability）：

成熟性（Maturity）：软件系统是否稳定、可靠，并且经过充分测试。

容错性（Fault Tolerance）：软件在面对故障或异常条件时的处理能力。

可恢复性（Recoverability）：软件在发生故障后能够恢复正常运行的能力。

可用性（Usability）：

易学性（Learnability）：用户能够快速学习和掌握软件的使用。

易操作性（Operability）：软件界面的友好程度和易用性。

用户体验（User Experience）：软件是否能够提供良好的用户体验。

效率（Efficiency）：

时间效率（Time Efficiency）：软件完成任务所需的时间。

空间效率（Space Efficiency）：软件在运行时所需的资源（如内存、存储空间）。

资源利用率（Resource Utilization）：软件在使用系统资源方面的效率。

可维护性（Maintainability）：

可读性（Readability）：软件代码是否易于阅读和理解。

可测试性（Testability）：软件是否易于进行测试。

可扩展性（Scalability）：软件是否易于扩展和修改。

适应性（Adaptability）：软件能否轻松地适应不同的环境和条件。

可移植性（Installability）：软件在不同平台上的安装和部署的便捷性。

替代性（Replaceability）：软件是否容易被替换或集成到其他系统中。

**基线建立的时间**是指在软件开发生命周期中确定和记录软件系统的基线（Baseline）的时间点。基线是软件开发过程中的一个重要里程碑，它代表了软件系统在特定时间点的稳定版本或状态。

基线建立的时间通常在软件开发的早期阶段，具体时间会根据项目的规模、复杂性和开发方法而有所差异。在软件开发过程中，建立基线的目的是为了标识和记录软件系统的初始状态，以便后续的开发、测试和变更控制。

**一般而言，在基线建立之前，需要完成以下任务：**

**需求分析和规划**：对软件系统的需求进行详细分析和规划，包括功能需求、性能需求、用户需求等。

**设计和架构：**基于需求分析结果，进行软件系统的设计和架构，确定软件系统的整体结构和组件。

**编码和单元测试**：根据设计和架构，进行编码实现，并进行单元测试来验证代码的正确性和功能性。

**集成和系统测试**：将各个组件进行集成，并进行系统测试来验证整体功能和性能。

**基线通常包含软件项目中的以下配置项：**

源代码：软件系统的源代码文件，包括各种编程语言的源代码文件。

可执行文件：编译或构建后生成的可执行文件，用于在特定平台上运行软件。

配置文件：用于配置软件系统行为和设置的文件，例如环境配置、数据库配置、日志配置等。

第三方库和依赖项：软件系统所依赖的第三方库、框架、组件或工具的版本信息。

数据库脚本：数据库的结构定义和初始化数据的脚本文件。

文档：软件系统的设计文档、用户手册、API文档、操作手册等。

测试用例和数据：用于软件系统测试的测试用例和测试数据。

部署脚本和说明：用于将软件系统部署到特定环境的脚本和说明文件。

版本控制配置：包括版本控制工具（例如Git）的配置文件和设置。

构建脚本和工具：用于构建、编译、打包和发布软件系统的脚本和工具。

**软件开发各阶段的配置基线如下：**

计划阶段：开发计划

需求分析阶段：需求规格说明、用户手册

设计阶段：设计规格说明

编码阶段：程序清单

测试阶段：测试报告