

软件项目开发过程

1 软件生命周期、软件过程模型的概念及关系

软件生命周期：把孕育、诞生、成长、成熟、衰亡的过程划分为若干阶段，每个阶段有明确的任务，不同阶段的特征、任务、产品、所需技术不一样。

软件过程模型（软件生命周期模型）：为软件工程技术提供了特定的路线图，该路线图规定了所有活动的流程、动作、任务、迭代的程度、工作产品及要完成的工作应如何组织。

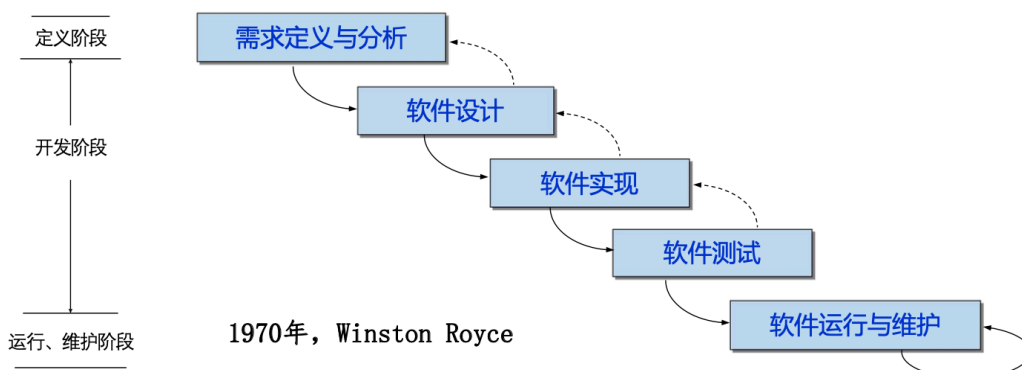
2 明白软件开发过程的典型阶段

计划 → 需求分析 → 软件设计 → 软件实现 → 软件验证 → 软件维护

- 计划：人们通过开展技术探索和市场调查等活动，研究系统的可行性和可能的解决方案，确定待开发系统的总体目标和范围
- 需求分析：在可行性研究之后，分析、整理和提炼所收集到的客户需求，建立完整的需求分析模型，编写软件需求规格说明
- 软件设计：根据需求规格说明，确定软件体系结构，进一步设计每个系统部件的实现算法、数据结构及其接口等
- 软件实现：概括地说是将软件设计转换成程序代码，这是一个复杂而迭代的过程，要求根据设计模型进行程序设计以及正确而高效地编写和测试代码
- 软件验证：检查和验证所开发的系统是否符合客户期望，包括单元测试、子系统测试、集成测试和验收测试等
- 软件维护：系统投入使用后对其进行改进，以适应不断变化的需求。完全从头开发的系统很少，将软件系统的开发和维护看成是一个连续过程更有意义

3 典型软件过程模型

3.1 瀑布模型



基本思想

- 将软件开发过程划分为分析、设计、编码、测试等阶段
- 工作以线性方式进行，上一阶段的输出是下一阶段的输入
- 每个阶段仅有里程碑和提交物

特点

- 需求最为重要，假设需求是稳定的
- 以文档为中心，文档是连接各阶段的关键

问题

- 文档多且完善——文档的评估需要各领域专家，文档是否有效，某一文档调整后对其他文档的影响，大量文档就一定有效吗，把用户隔离在开发过程之外
- 瀑布模型要求客户明确需求，但实际中客户在看到软件前无法可靠地描述他们想要什么
- 客户必须要有耐心，因为在项目接近尾声时才能得到可执行的程序
- 容易导致阻塞状态，因为任务之间具有依赖性

适用场合

- 软件项目较小
- 需求在项目开始之前已经被全面了解
- 需求在开发中不太可能发生重大改变
- 外部环境的不可控因素很少

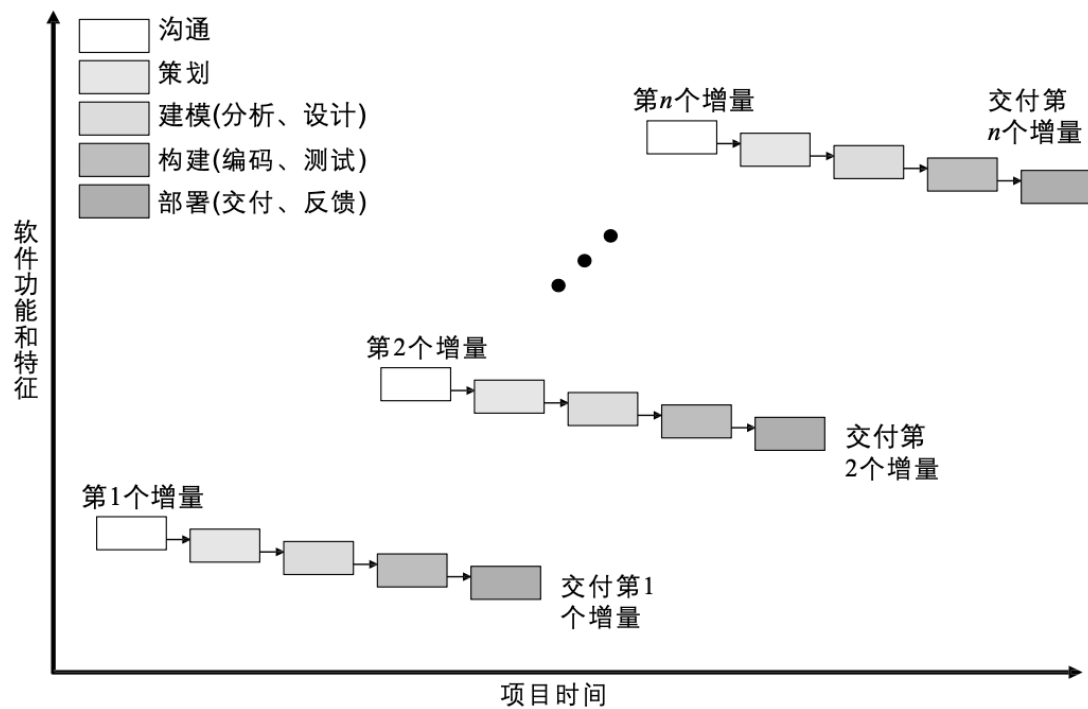
优点

- 简单、易懂、易用、快速
- 项目管理容易
- 每个阶段必须提供文档，必须进行审查，可操作性强

缺点

- 难以快速响应用户需求变化
- 不容易满足客户需求
- 客户必须在项目接近尾声的时候才能得到可执行的程序，对系统中存在的重大缺陷，如果在评审之前没有被发现，将可能会造成重大损失

3.2 增量过程模型——增量模型



使用方法：软件被作为一系列的增量来进行开发，每一个增量都提交一个可以操作的产品，可供用户评估

- 第一个增量往往是核心产品：满足了基本的需求，但是缺少附加的特性
- 客户使用上一个增量的提交物并进行评价，制定下一个增量计划，说明需要增加的特性和功能
- 重复上述过程，直到最终产品产生为止
- 本质是以迭代的方式运用瀑布模型

优点

- 在时间要求较高的情况下交付产品：在各个阶段并不交付一个可运行的完整产品，而是交付满足客户需求的一个子集的可运行产品，对客户起到“镇静剂”的作用
- 人员分配灵活：如果找不到足够的开发人员，可采用增量模型，早期的增量由少量人员实现，如果客户反响较好，则在下一个增量中投入更多的人力
- 逐步增加产品功能可以使用户有较充裕的时间来学习和适应新产品，避免全新软件可能带来的冲击
- 因为具有较高优先权的模块被首先交付，而后面的增量也不断被集成进来，这使得最重要的功能肯定接受了最多的测试，从而使得项目总体性失败的风险比较低

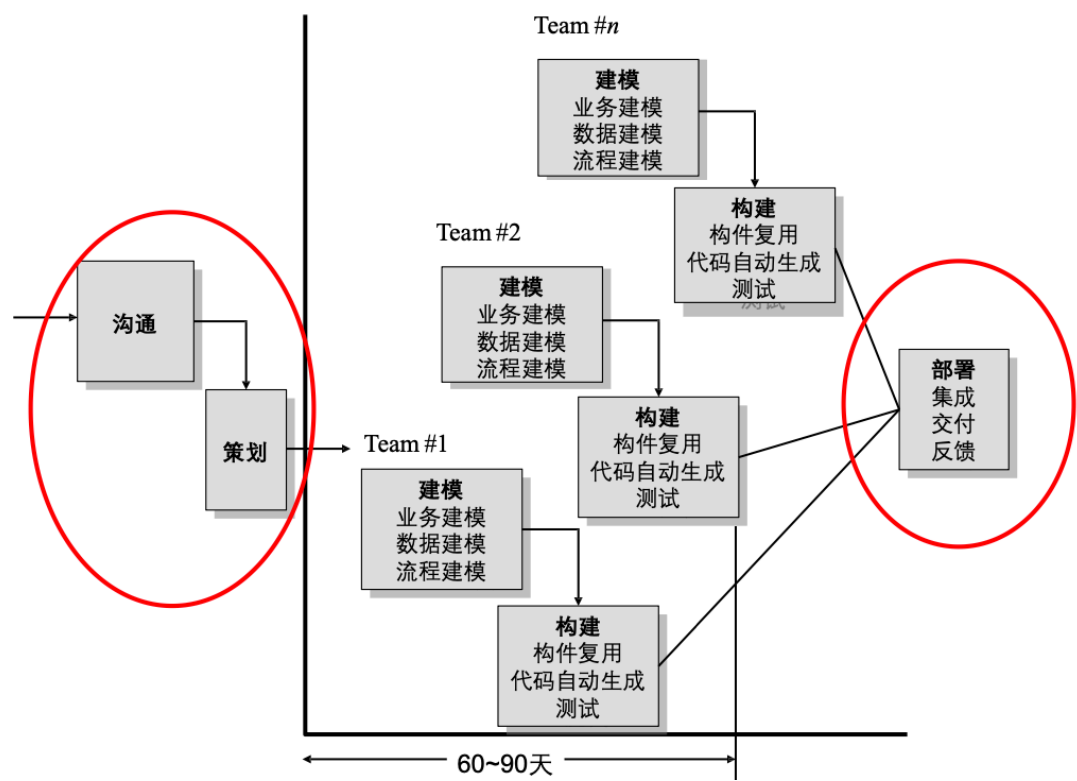
缺点

- 每个附加的增量并入现有的软件时，必须不破坏原来已构造好的东西
- 同时，加入新增量时应简单、方便——该类软件的体系结构应当是开放的

适用场景

- 在开始开发时，需求很明确，且产品还可被适当地分解为一些独立的、可交付的软件
- 在开发中，期望尽快提交其中的一些增量产品

3.3 增量过程模型——快速应用程序开发(RAD)



本质是瀑布模型的高速变体，并行运行瀑布模型

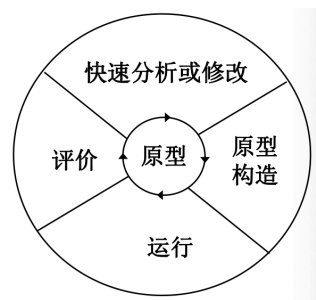
优点

- 提高软件交付速度
- 充分利用企业已有资产进行项目开发

缺点

- 需求充分理解，系统被合理的模块化
- 需要大量的人力资源来创建多个相对独立的 RAD 团队
- 要求管理水平高，如果没有在短时间内为急速完成整个系统做好准备，RAD 项目将会失败
- 如果系统需求是高性能，并且需要通过调整构件接口的方式来提高性能，不能采用 RAD 模型
- 技术风险很高的情况下（采用很多新技术、软件需与其他已有软件建立集成、等等），不宜采用 RAD

3.4 演化过程模型——快速原型开发模型



快速原型是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。

原型在开发中的作用

- 获得用户的真正需求
- 可用于为一个项目或项目中某些部分，确定技术、成本和进度的可能性

步骤：

- 原型快速分析：指在分析者和用户的紧密配合下，快速确定软件系统的基本要求
- 原型构造：在原型分析的基础上，根据基本需求规格说明，忽略细节，只考虑主要特性，快速构造一个可运行的系统
- 原型运行与评价：软件开发人员与用户频繁通信、发现问题、消除误解的重要阶段，目的是发现新需求并修改原有需求
- 原型修正：对原型系统，要根据修改意见进行修正
- 判定原型完成：如果原型经过修正或改进，获得了参与者的一致认可，那么原型开发的迭代过程可以结束

对比增量模型：

增量模型：先构造一个核心功能，往里面加东西，每次是一个可操作软件，最后是产品的一个部分。

原型开发：得到基本需求后，简单分析就开始开发。用户不满意，原型有可能抛弃。原型是让用户来拿来进行评估和提意见的，可以是用户界面，或某一阶段的文档。根据用户的意见再完善。

优点

- 快速开发出可以演示的系统，方便与客户沟通
- 采用迭代技术能够使开发者逐步弄清客户的需求

缺点

- 为了尽快完成原型，开发者没有考虑整体软件的质量和长期的可维护性，系统结构通常较差
- 用户可能混淆原型系统与最终系统，原型系统在完全满足用户需求之后可能会被直接交付给客户使用

3.5 演化过程模型——螺旋模型



在瀑布模型和演化模型的基础上，加入两者所忽略的风险分析所建立的一种软件开发模型

该模型将软件生存周期的活动分为四个可重复的阶段：规划、风险分析、开发和评估

步骤：

- 评估和风险分析阶段都可作出一个决策：项目是否继续
- 螺旋循环的次数指示了已消耗的资源

- 在规划阶段、风险分析阶段和开发阶段均进行需求规约活动
- 在早期螺旋循环中，为了为最终的实现给出一些指导性决策，经常使用原型构造
- 设计和实现活动一般是在开发阶段进行

优点：结合了原型的迭代性质与瀑布模型的系统性和可控性，是一种风险驱动型的过程模型

- 采用循环的方式逐步加深系统定义和实现的深度，同时更好的理解、应对和降低风险
- 确定一系列里程碑，确保各方都得到可行的系统解决方案
- 始终保持可操作性，直到软件生命周期的结束
- 由风险驱动，支持现有软件的复用

缺点

- 适用于大规模软件项目，特别是内部项目，周期长、成本高
- 软件开发人员应该擅长寻找可能的风险，准确的分析风险，否则将会带来更大的风险
- 由于构建产品所需的周期数据不确定，给项目管理带来困难
- 演化速度不易把握，演化速度太快，项目陷入混乱；演化速度太慢，影响生产率
- 为追求软件的高质量而牺牲了开发速度、灵活性和可扩展性

4 对比几种典型的软件过程模型的异同

选择模型时考虑以下维度：时间效率、成本、人力资源、开发质量、顾客满意度、需求扩展、需求变化、风险、与顾客交互程度、适用项目规模、适用 deadline 紧急程度、项目管理的方便程度、等等。

软件项目开发管理

1 软件项目管理概念及特征

项目(Project): 精心定义的一组活动, 使用受约束的资源(资金、人、原料、能源、空间等)来满足预定义的目标

项目管理(Project Management, PM): 有效的组织与管理各类资源(例如人), 以使项目能够在预定的范围、质量、时间和成本等约束条件下顺利交付(deliver)

软件项目的**特征**:

- 软件产品的不可见性
- 项目的高度不确定性
- 软件过程的多变化性
- 软件人员的高技能及高流动性

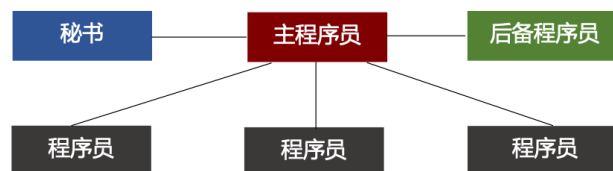
软件项目管理的 **4P**:

- **People**: 软件人员

需求分析员、架构师、程序员、测试人员、培训人员.....

团队的组织方式:

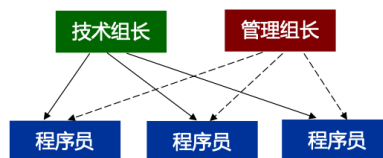
- **主程序员式组织结构**: 以主程序员为核心, 主程序员既是项目管理者也是技术负责人, 团队其他人员的职能进行专业化分工。



优点: 成员之间采取简单的交流沟通模式






缺点: 很难找到技术和管理才能兼备的主程序员

- **矩阵式组织结构**: 技术管理分离, 技术负责人负责技术决策, 管理负责人负责非技术性事务的管理决策和绩效评价。



在这种组织中, 明确划分技术负责人和管理负责人的权限是十分重要的。

- **Product**: 软件产品

需求分析	软件设计	软件实现	软件测试	软件运行
 <ul style="list-style-type: none"> • 用例模型 • 软件需求规格说明 	 <ul style="list-style-type: none"> • 软件体系结构描述 • 设计模型 	 <ul style="list-style-type: none"> • 源程序 • 目标代码 • 可执行构件 	 <ul style="list-style-type: none"> • 测试规程 • 测试用例 	 <ul style="list-style-type: none"> • 相关的运行时文件 • 用户手册
<div>开发管理文档</div> <div> <div>计划文档</div> <ul style="list-style-type: none"> - 工作分解结构 - 业务案例 - 发布规格说明 - 软件开发计划 </div> <div> <div>操作文档</div> <ul style="list-style-type: none"> - 发布版本说明书 - 状态评估 - 软件变更申请 - 实施文档、环境 </div>				

● **Process:** 软件过程

1. 选择软件过程模型
2. 基于过程框架制定初步的项目计划
3. 过程分解，制定完整计划，确定工作任务列表，任务对应产出物

● **Project:** 项目

W⁵HH 原则：通过提出一系列问题，来导出对关键项目特性和项目计划的定义：Why, What, When, Who, Where, How, How much

基本要素：结果，工作，进度表，资源

软件项目管理计划：一个用来协调所有其他计划、以指导项目实施和控制的文件，它应该随着项目的进展和信息的补充进行定期完善

2 软件项目估算

项目估算是对项目的规模、工作量、时间和成本等进行预算和估计的过程。

项目估算内容：规模估算、工作量估算、进度估算、成本估算

估算的复杂不确定性：估算的风险取决于资源、成本及进度的定量估算中存在的 uncertainty。

基本估算方法：分段估算，专家判断，参数估算

分段估算：

从宏观估算开始，在项目执行中对各阶段的估算进行细化；适用于最终产品不可知或不确定性很大的项目。

专家判断：

通过借鉴历史信息，专家提供项目估算所需的信息，或根据以往类似项目的经验，给出相关参数的估算上限。

参数估算：

通过对大量的项目历史数据进行统计分析，使用项目特性参数建立经验估算模型，估算诸如成本、预算和持续时间等活动参数。



• 代码行技术 (LOC)

$$L = \frac{a + 4m + b}{6} \quad C = \mu \times L \quad PM = \frac{L}{v}$$

L : 估计的代码行数, a : 乐观值, b : 悲观值, m : 可能值;

C : 总成本, μ : 每行代码的单位成本;

PM : 总工作量 (人月), v : 平均生产率 (代码行数/人月)

• 功能点技术方法 FP

功能点方法是依据软件信息域的基本特征和对软件复杂性的估计, 估算出软件规模。这种方法适合于在软件开发初期进行估算, 并以功能点为单位度量软件规模。

功能点估算使用 5 种信息域:

- 外部输入: 通过界面等的输入, 插入和更新等操作都是典型外部输入
- 外部输出: 仅仅是输出, 例如导出、报表、打印等
- 外部查询: 先输入数据, 再根据输入数据计算得到输出, 例如查询
- 内部逻辑文件: 可以理解为业务对象, 可能对应多个数据表
- 外部接口文件: 其它应用提供的接口数据

估算得到一个系统的总功能点数, 然后据此估算工作量和成本。

信息域加权因子:

信息域参数	加权因子			合计
	简单	中等	复杂	
外部输入	3	4	6	Σ
外部输出	4	5	7	Σ
外部查询	3	4	6	Σ
内部逻辑文件	7	10	15	Σ
外部逻辑文件	5	7	10	Σ
未调整功能点 UFC				Σ

系统复杂度调整值 F_i : 取值 0-5

F_1	可靠的备份和恢复	F_8	在线升级
F_2	数据通信	F_9	复杂的界面
F_3	分布式处理	F_{10}	复杂的数据处理
F_4	性能	F_{11}	代码复用性
F_5	大量使用的配置	F_{12}	安装简易性
F_6	联机数据输入	F_{13}	多重站点
F_7	操作简单性	F_{14}	易于修改

$$\text{功能点计算: } FP = UFC \times [0.65 + 0.01 \times \Sigma F_i]$$

• COCOMO 模型 (结构性成本模型)

一种利用经验模型进行成本估算的方法

$$PM_{nominal} = A \times (Size)^B$$

$PM_{nominal}$: 人月工作量, A : 工作量调整因子, B : 规模调整因子, $Size$: 规模, 单位是千行代码或功能点数。

类型	A	B	说明
组织型	2.4	1.05	相对小的团队在一个高度熟悉的内部环境中开发规模较小, 接口需求较灵活的系统。
嵌入型	3.6	1.2	开发的产品在高度约束的条件下进行, 对系统改变的成本很高。
半独立型	3.0	1.12	介于上述两者中间

● 用例点估算

	基本流	扩展流	业务规则	折合标准用例
功能A	12	3	4	2.3
功能B	8	4	3	1.8
功能C	6	2	3	1.4
标准用例 = (基本流+扩展流+2×业务规则) / 10 生产率 = 6工作日 / 单位用例 工作量 = 6×5.5 = 33工作日				合计 5.5

● 故事点方法

故事点: 表达用户故事、功能或其他工作的总体规模的度量单位, 它是一个**相对度量单位**。使用时可以给每个故事分配一个点值, 点值本身并不重要, 重要的是点值的相对大小。

理想日: 表达用户故事、功能或其他工作的总体规模的另外一种度量单位, 它是一个**绝对度量单位**。理想时间是某件事在剔除所有外围活动以后所需的时间; 一般为一天有效工作时间的 60-80% 比较合理, 但绝不会是全部。

故事点的基本做法: 把一些常见“标准任务”给出一个“标准点数”, 形成比较基线; 估算时只要是同一类型任务, 直接写故事点数而非天数。

● 机器学习方法

人工神经网络

基于案例的推理方法: 识别出与新项目类似的案例, 再调整这些案例, 使其适合新项目的参数。

3 项目进度安排

软件延期交付的原因: 不切实际的开发日期; 客户需求变更; 对完成该工作量所需资源估计不足; 出现了无法应付的技术难题; 出现了无法预计的人力问题; 由于项目团队成员之间的交流不畅而导致的延期; 项目管理者未发现项目进度拖后, 也未能采取措施解决这一问题.....

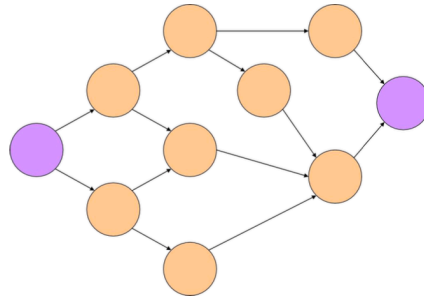
软件项目进度计划:

1. 将项目划分为多个活动、任务。**PBS、WBS**

40-20-40 法则：40%工作量分配给前期的分析和设计；20%工作量分配给编码；40%分配给测试

2. 确定任务项目依赖性。**定义任务网络**

各任务在顺序上存在依赖关系，网络图

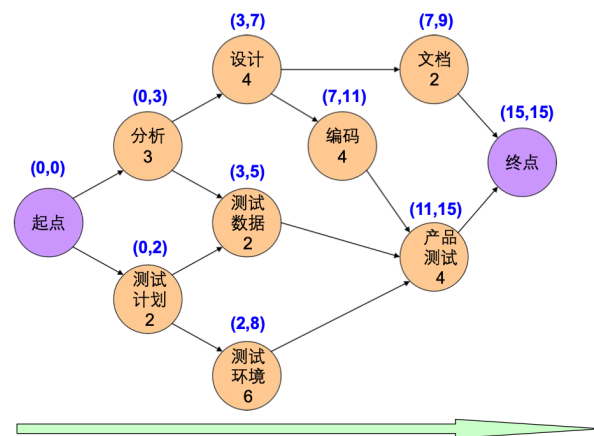


3. 时间分配。**为任务安排工作时间段**

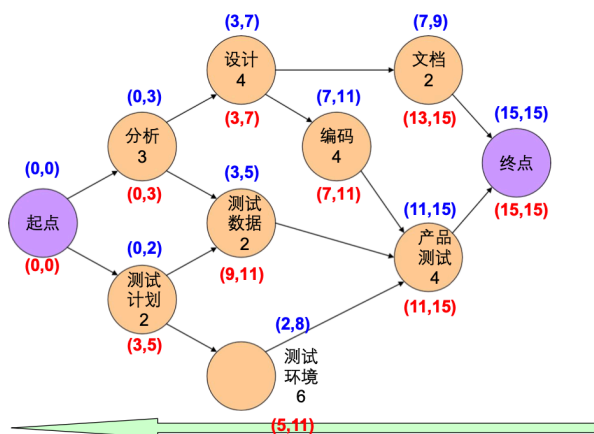
两种具体技术：程序评估及评审技术(PERT)；关键路径方法(CPM)

步骤：

(a) 标出任务最早开始时间和结束时间



(b) 标出任务最晚开始时间和结束时间



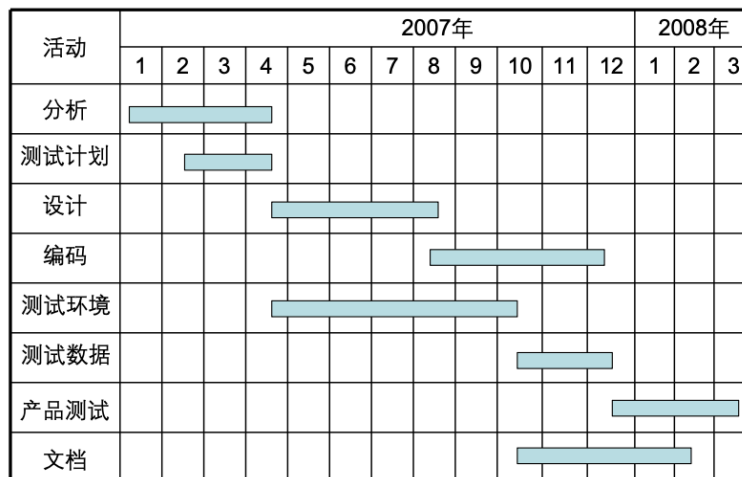
(c) 计算关键路径

最早开始/结束时间=最晚开始/结束时间的点

(d) 确定任务的开始/结束时间

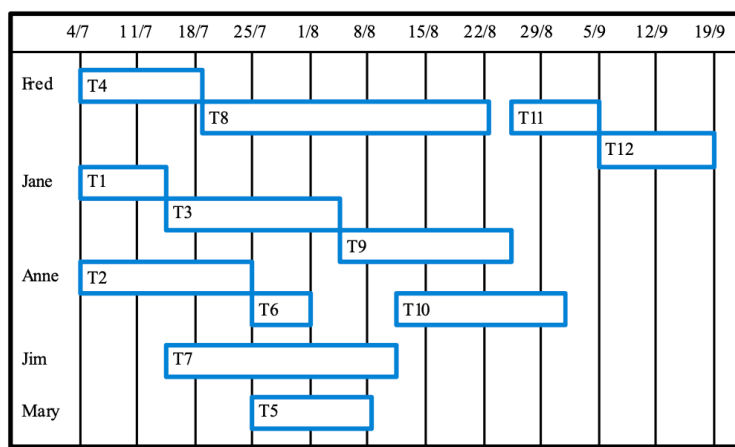
先确定关键路径上各任务的日期，然后确定其他任务的日期；通过缩短关键路径上的任务的持续时间，可极大优化整个项目的持续时间。

(e) 绘制最终的任务进度安排（甘特图）

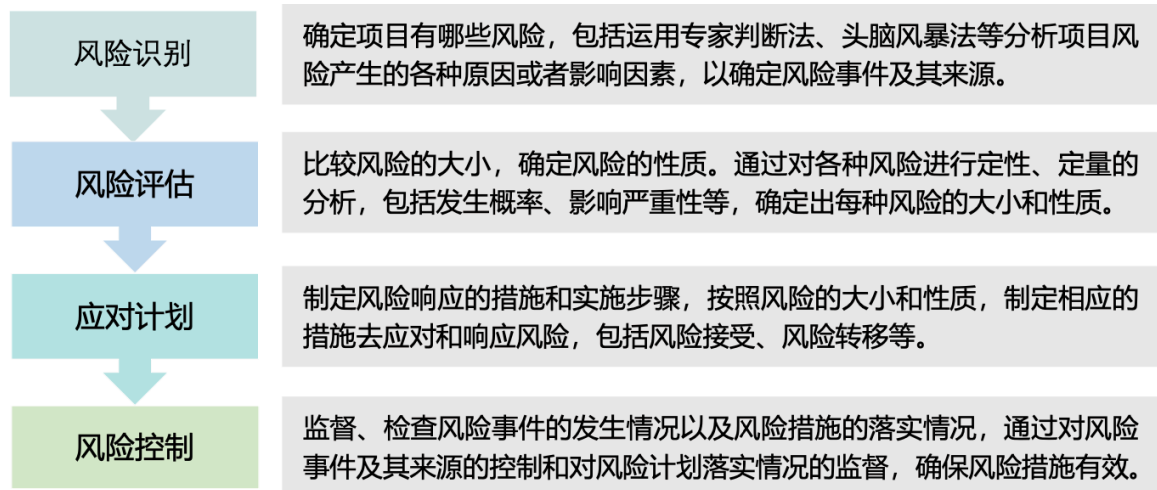


4. 确认任务资源。确认每个任务的资源需求
5. 确定责任。确定每个任务的负责人和参与人
6. 明确结果。明确任务的输出结果(文档或部分软件)
7. 确定里程碑(milestones)。确定任务与项目里程碑关联

人员/资源分配图：



4 项目风险管理



风险管理的 7 个原则

- 保持全面观点：考虑软件风险及软件所要解决的任务问题
- 采用长远观点：考虑将来要发生的风险，并制定应急计划使将来发生 的风险成为可管理的
- 鼓励广泛交流：如果有人提出一个潜在的风险，要重视它
- 结合：考虑风险时必须与软件过程相结合
- 强调持续的过程：整个软件过程中，要持续保持警惕。随着信息量增加，要修改已识别的风险；随着知识的增加，要加入新的风险
- 开发共享的产品：如果所有利益相关者共享相同版本的软件产品，有利于风险识别和评估
- 鼓励协同工作：汇聚利益相关者的智慧、技能和知识