

2-3 软件项目管理

2019年4月13日 16:22

[1 一个小案例](#)

[2 人员](#)

[3 产品](#)

[4 过程](#)

[5 项目](#)

[6 可行性分析与估算](#)

[7 项目进度计划与监控](#)

若干基本概念

- 项目(Project):为创建某种特定的产品或服务而组织或设计的临时的、一次性的行动;通过执行一组活动,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(ProjectManagement,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)。
 - 挑战1: 在各类约束条件下交付项目;
 - 挑战2: 通过优化资源的分配与集成来满足预先定义的目标;

软件项目的特征

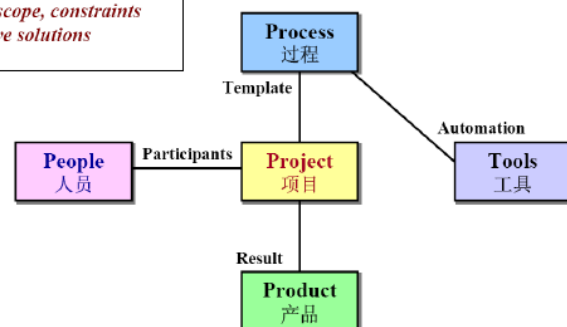
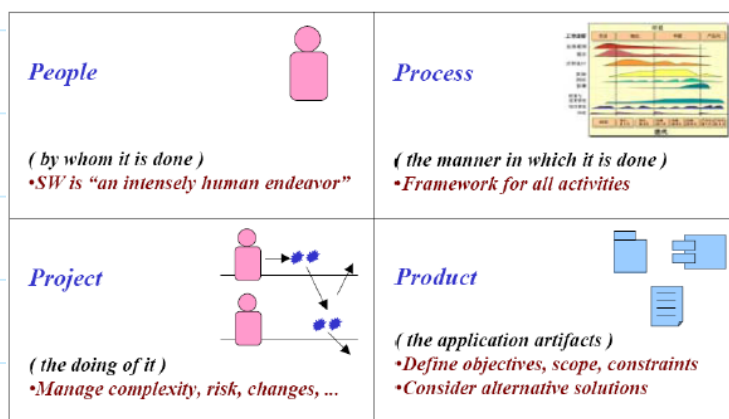
- 软件产品的不可见性->软件项目复杂和抽象
开发过程和产品都是看不见摸不着的,导致软件项目特别复杂和抽象
- 项目的高度不确定性->预定计划于实际情况存在较大偏差
项目的估算与计划非常困难,有很多难以预见的问题,造成预定计划于实际情况存在较大偏差;
- 软件过程的多变化性->不确定、不稳定
迭代、增量开发、动态变化、不确定、不稳定;
- 软件人员的高技能及其高流动性->风险
智力密集型活动、对人的要求高、核心技术人才流动性高。

1 一个小案例

- X项目的初始状态
 - 一个年轻的项目经理A
 - 10个经验缺乏的技术人员
 - 项目交付期:紧张

- 技术风险：较高
- X项目的当前状态
 - 进度已经落后于计划
 - 项目经理A已经向客户汇报了一次项目开发进度，并已经演示了系统功能，已开发的功能已被用户接受并认可。
 - 此时：项目组补充加入了一位水平高、经验丰富的技术人员B
 - B检查了团队目前完成的代码，发现：原来写的代码效率不高，构架繁冗，不方便后期维护，也可能导致软件性能方面存在重大缺陷。
 - B的建议：重构代码和数据库设计。
 - 团队的想法：辛苦写的代码被一票否决，心里很不舒服，但是也承认自己的代码质量不高。
 - 项目经理A的想法：预计客户将来会提到这个问题，而届时再重构，会更加麻烦。
 - 公司CEO的意见：已经向客户申请过一次计划调整并基本得到用户的理解，但目前进度已经落后于计划，急需完成剩余部分功能的开发。不能再次申请延期。
- X项目的客观情况
 - 如果系统重构，很有可能需要再次调整计划，而用户已经明确表示计划是不可能再调整的了。
 - B在技术上没有问题，但是缺乏时间观念，且身兼多个项目，重构速度令人失望。
 - 老成员一开始挺配合系统重构，但是随着重构的深入，发现难度很大，基本等于重新开发。于是对B很有意见，不怎么配合B的指挥。
- 如果你作为项目经理，怎么办？

软件项目管理的“4P”



2 软件项目的参与人员(People)

- 高级管理者(项目经理/产品经理)：负责定义业务问题；

- 项目(技术)管理者(项目经理/产品经理): 计划、激励、组织和控制软件开发人员;
- 开发人员: 拥有开发产品或应用软件所需技能的人员;
- 系统分析员; 系统架构师; 设计师; 程序员; 测试人员; 质量保证人员
- 客户: 进行投资、详细描述待开发软件需求、关心项目成败的组织/人员
- 最终用户: 一旦软件发布成为产品, 最终用户就是直接使用软件的人。

项目经理(ProjectManager)

- 最重要的: 领导力 (MOI模型)
 - Motivation(激励): 通过“推”或“拉”鼓励项目成员发挥其最大才能与潜力;
 - Organization(组织): 形成能够将最初概念转换为最终产品的能力;
 - IdeatorInnovation(思想或创新): 即使在诸多约束条件下工作, 也能鼓励项目成员去创造新的想法。
- 项目经理的关键品质
 - 解决问题: 准确诊断出关键技术问题、组织问题, 制定解决方案
 - 管理者特性: 掌控整个项目, 具有领导力
 - 成就: 通过激励优化团队效率
 - 影响和队伍建设: 理解“人”, 在高压环境下保持控制力

软件开发团队

- “最好的”团队取决于项目经理的管理风格、团队里的人员数目与技能水平、项目的总体难易程度;
- 组建团队时应考虑以下要素:

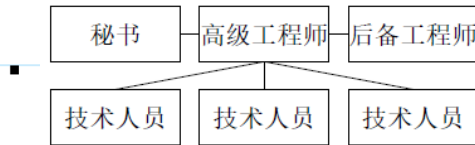
| | |
|---|---|
| <ul style="list-style-type: none"> – 从项目需求来看: <ul style="list-style-type: none"> • 待解决问题的难度; • 待开发软件系统的规模; • 待开发软件系统的技能要求; • 交付日期的严格程度; ◦ 共同工作的时间; • 彼此之间的人际关系与友好交际程度; • | <ul style="list-style-type: none"> – 从个人能力来看: <ul style="list-style-type: none"> • 应用领域经验 • 开发平台经验 • 编程经验 • 教育背景 • 沟通能力 • 适应能力 • 工作态度 • 团队协作能力 • |
|---|---|

软件开发团队的组织方式

- 一窝蜂模式(chaosteam): 没有明确分工, 存活的时间一般都不长。
- 主治医师模式:(Chief-ProgrammerTeam,surgicalteam)
 - 手术台上, 有一个主刀医师, 其他人(麻醉、护士、器械)各司其职, 为主刀医师服务。
 - 首席程序员(Chief-programmer)处理主要模块的设计和编码, 其他成员从各种角度支持他的

工作(backupprogrammer,admin,tool-smith,specialist)。

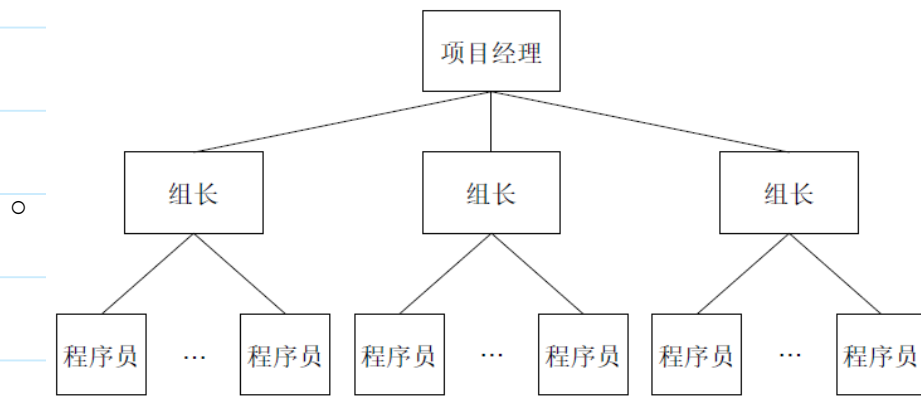
- 主治医师模式的退化：学校里，软件工程的团队模式往往退化为“一个学生干活，其余学生跟着打酱油”。
- 明星模式(Super-starmodel)



- 社区模式(CommunityModel):
 - 由很多志愿者参与，每个人参与自己感兴趣的项目，贡献力量。
 - 好处是“众人拾柴火焰高”，但是如果大家都只来烤火，不去拾柴；或者捡到的柴火质量太差，最后火也熄灭了。
 - “社区”并不意味着“随意”，一些成功的社区项目(例如开发和维护Linux操作系统的社区)都有很严格的代码复审和签入的质量控制。
 - 开源社区(OpenSourceProject)
- 交响乐团模式(Orchestra)
 - 人多，门类齐全，各司其职，各自有专门场地，演奏期间严格遵循纪律。
 - 演奏靠指挥协调，各自遵循曲谱(工作流程)；
 - 演奏的都是练习过多次的曲目，重在执行。
 - 类似于“工厂”，严格遵循预订的生产流程，“规格严格”
- 爵士乐模式(JazzBand)
 - 演奏时没有谱子，没有现场指挥，平时有arranger起到协调和指导作用
 - 模式：主乐手先吹出主题，其余人员根据这个主题各自即兴发挥；主乐手最后再加入，回应主题，像是对曲子的总结
 - “强调个性化的表达，强有力的互动,对变化的内容有创意的回应”
 - 类似于一群天才构成的敏捷团队，“功夫到家”，率性而为。
- 功能团队模式(featureteam)
 - 具备不同能力的同事平等协作，共同完成一个项目开发；
 - 在这个项目完成之后，这些人又重新组织，和别的角色一起去完成下一个功能，他们之间没有管理和被管理的关系。
- 官僚模式(bureaucraticmodel)
 - 成员之间不光有技术方面的合作和领导，同时还混进了组织上的领导和被领导关系，跨组织的合作变得比较困难。

组织分解结构(OBS)

- 项目管理里通常使用“组织结构分解(OrganizationBreakdownStructure,OBS)”作为描述组织/人员之间关系的工具：



敏捷团队



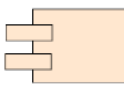
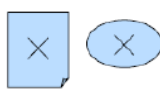

- 小型充满活力的团队
- 强调团队成员的个人能力与团队协作精神相结合
- 自组织团队
 - 项目管理自主权
 - 技术决定权
 - 计划制定工作压缩到最低
 - 团队选择自己适用的手段（过程、方法、工具）

人员协调与沟通

- 问题1：为什么需要沟通？
- 问题2：沟通的方式有哪些？
 - 面对面交谈、电话交谈、email、面对面会议、电话会议、网络会议、项目网站、书面报告；
- 问题3：项目沟通活动有哪些？
 - 规划项目沟通；
 - 实施阶段性评审；
 - 每周小组会议；

3 产品(Product)

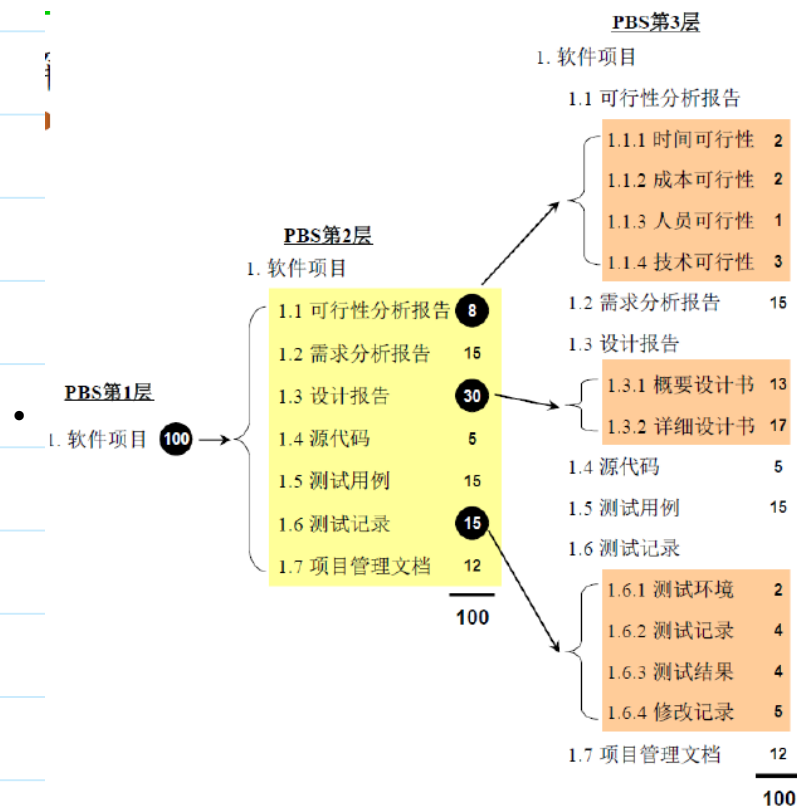
软件产品

| 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<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>操作文档<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></div> | | | | |

软件产品、产品分解结构(PBS)

- 首先应确定软件范围：
 - 功能和非功能(性能)
 - 在管理层和技术层都必须是无歧义的和可理解的，软件范围应是确定的；
- 一旦确定了范围，需要对其进行分解——分而治之。
- 项目管理里通常使用“产品结构分解(Product Breakdown Structure, PBS)”作为产品分解的工具：
 - PBS：通过分层的树型结构来定义和组织项目范围内的所有产出物(产品)，自顶向下，逐级细分；
 - 产出物：项目结束时需要提交的最终产品，在项目之初就可以准确的预计。

产品分解结构(PBS)



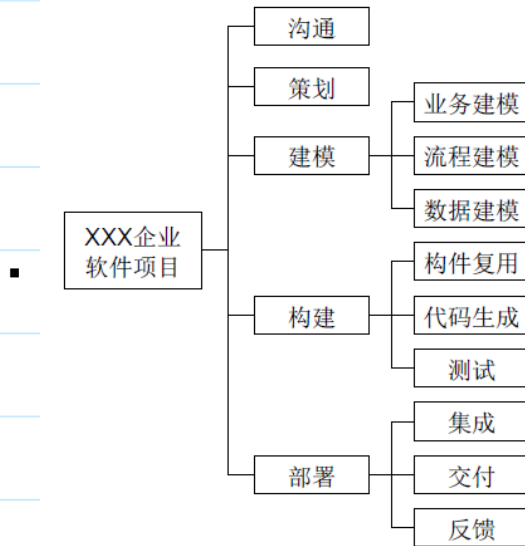
4 过程(Process)

软件过程

- Step1:选择合适的软件过程模型
 - 存在多种过程模型
 - 各过程模型适用不同类型的软件项目
- Step2:根据所选的过程模型，对其进行适应性修改；
- Step3:确定过程中应包含的工作任务列表；
 - [例]沟通活动：
 - 列出需澄清的问题清单；
 - 与客户见面并说明问题；
 - 共同给出范围陈述
 - 与所有相关人员一起评审；
 - 根据需要修改范围陈述。

工作分解结构(WBS)

- 项目管理里通常使用“工作结构分解(WorkBreakdownStructure,WBS)”作为过程分解的工具。
- WBS: 通过分层的树型结构来定义和组织工作任务之间的分解关系，自顶向下，逐级细分；
 - [例]RAD过程模型的WBS分解结构



5 项目(Project)

- 项目关注的四个方面
 - 范围(Scope)
 - 时间(Time)
 - 成本(Cost)
 - 质量(Quality)
- 项目管理的主要任务
 - 项目可行性分析与估算
 - 项目进度安排
 - 项目风险管理
 - 项目质量管理
 - 项目跟踪与控制
- 如何定义关键的项目特性--W5HH原则
 - Why为什么要开发这个系统?
 - What将要做什么?
 - When什么时候做?
 - Who某功能由谁来做?
 - Where他们的机构组织位于何处?
 - How如何完成技术与管理工作?
 - Howmuch各种资源分别需要多少?
- 项目管理中的危险信号
 - 软件人员不了解其客户的要求
 - 产品范围定义的很糟糕
 - 没有很好地管理变更

- 选择的技术发生了变化
- 业务需求发生变化（或未被很好地定义）
- 最后期限是不切实际的
- 客户抵制
- 失去赞助（或从来没有真正得到过赞助）
- 项目团队缺乏具有合适技能的人员
- 管理者（和开发人员）没有很好地利用已学到的最佳实践和经验
- 90—90规则

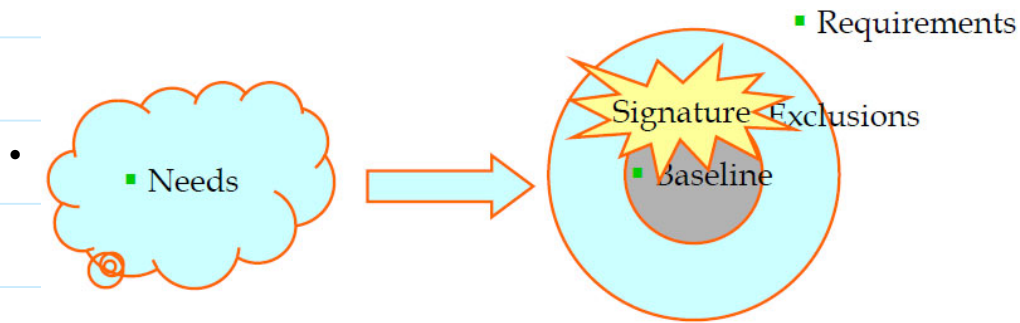
6 可行性分析与估算

估算与可行性分析

- 在项目开始之前，必须预先估计三件事情（定量）：
 - 需要多少工作量（成本）
 - 需要多少时间（进度）
 - 需要多少人员、设备（资源）
- 软件估算的依据
 - 历史信息
 - 经验
 - 定量预言
- 此外，还必须预测所需要的资源(硬件和软件)以及蕴含的风险；
 - 定量的不确定性
 - 资源不稳定性
- 从而得出“该项目是否可行”的结论。

确定范围

- 范围(Scope)：描述了将要交付给最终用户的功能和特性、输入输出数据、用户界面、系统的性能、约束条件、接口和可靠性等，以及期望的时间、成本目标；
- 两种方法：
 - 与所有项目成员交流之后，写出对软件范围的叙述性描述；
 - 由最终用户开发一组用例。
- 注意
 - 并不是客户所有的要求都“来者不拒”，需要分别对待！
 - 软件范围一定要用户同意，签字认可！



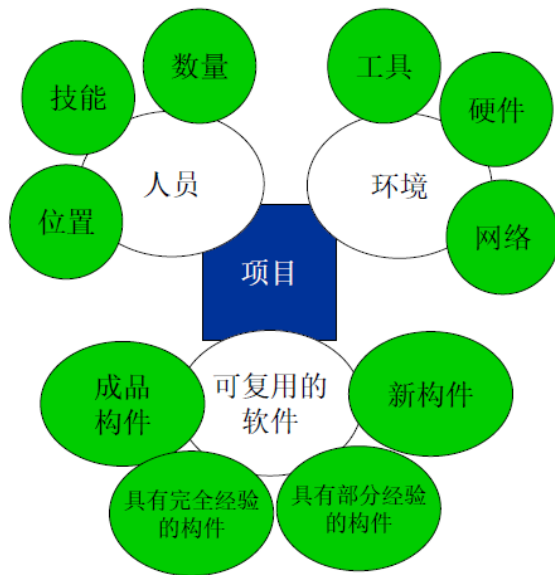
- Needs:客户/最终用户的请求、想法和业务需求;
- Requirements: 对未来系统所应具备的功能的陈述;
- Exclusions:将不包含在未来系统中的功能的陈述;
- Baseline:对未来系统中应包含的功能的陈述。

可行性分析

- 技术可行性：
 - 项目在技术上可行吗？它在技术水平范围内吗？能够将缺陷减少到一定程度吗？
- 经济可行性：
 - 它在经济上可行吗？能以可负担的成本完成开发吗？
- 时间可行性：
 - 项目投入市场的时间可以按预期完成吗？
- 资源可行性：
 - 组织拥有取得成功所需要的资源吗？

确定资源

- 资源
 - 人力资源
 - 可复用的软件资源
 - 环境资源
- 说明资源的特征
 - 资源的描述
 - 可用性说明
 - 何时需要资源
 - 使用资源的持续时间



软件项目估算

- 如何估算时间、成本、资源？—靠经验？—靠数学公式？
- 软件项目估算方式
 - 把估算推迟到项目的后期进行X
 - 根据已经完成的类似项目进行估算
 - 使用比较简单的分解技术，生成项目的成本和工作量的估算
 - 使用一个或多个经验模型来进行软件成本和工作量的估算
- 到目前为止，因为软件项目变化的要素太多，所以对软件的成本估算从来没有达到精确。但是，估计得越精确，项目成功的可能性就越高
- 方法：
 - 代码行技术
 - 功能点技术
 - 过程估算技术

分解估算技术

- 软件规模估算方法
 - 直接测量（如：代码行LOC）
 - 间接测量（如：功能点FP）
- 问题分解与过程分解（产品与过程）
 - 基于问题的估算
 - 将软件分解成可独立估算的功能问题（功能、类、变更、过程）
 - 估算每个功能的LOC或FP
 - 合并得到项目的总体估算
 - 基于过程的估算

- 调和不同的估算方法

代码行技术(LOC)LinesofCode

- 从过去开发类似产品的经验和历史数据出发，估计出所开发软件的代码行数。

$$L = \frac{a + 4m + b}{6}$$

L 估计的代码行数

a 乐观值

b 悲观值

m 可能值

○

$$C = \mu \times L$$

L 估计的代码行数

μ 每行代码的单位成本

C 总成本

$$PM = \frac{L}{v}$$

L 估计的代码行数

v 平均生产率(LOC/pm)

PM 总的工作量(pm)

- LOC示例：CAD软件包的开发

| 功能 | LOC估算 | | | |
|-------------|----------------------------|------|------|--------|
| | 乐观值 | 可能值 | 悲观值 | 估算值 |
| 用户接口及控制设备 | 2000 | 2200 | 3000 | 2300 |
| 二维几何分析 | 3000 | 5700 | 6000 | 5300 |
| 三维几何分析 | 4600 | 6900 | 8600 | 6800 |
| 数据库管理 | 3000 | 3250 | 4100 | 3350 |
| 图形显示 | 4000 | 4900 | 6100 | 4950 |
| 外部设备控制 | 1800 | 2100 | 2400 | 2100 |
| 设计分析模块 | 7000 | 8600 | 9000 | 8400 |
| 总代码行数(L) | | | | 33200 |
| 平均生产率(v) | 620LOC/人月(统计值) | | | |
| 每行代码单位成本(u) | 月平均工资8000, $u=8000/v=13$ 元 | | | |
| 总成本(C) | $C=u \times L$ | | | 431600 |
| 总工作量(PM) | $PM=L/v$ | | | 54人月 |

功能点技术FP

- 功能点(FunctionPoint,FP)，以功能点为单位来估计软件规模，关注五个方面的功能：
 - 外部输入(EI)：用户进行添加或修改数据的UI
 - 外部输出(EO)：软件为用户产生的输出UI
 - 外部查询(EQ)：软件可产生的独立查询
 - 内部逻辑文件(ILF)：软件修改或保存的逻辑记录集合(数据表或文件)
 - 外部接口(EIF)：与其它系统进行信息交换或共享的文件
- 问题分解关注的不是软件功能，而是信息域的值（输入和输出数量）
- 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。

Step1: 计算未调整功能点

| 信息域值 | 乐观值 | 可能值 | 悲观值 | 估算值 (P) | 加权因子 (r) | | | FP值 |
|----------|-----|-----|-----|------------|----------|----|----|-----|
| | | | | | 简单 | 中等 | 复杂 | |
| 外部输入 | 20 | 24 | 30 | 24 | 3 | 4 | 6 | 97 |
| 外部输出 | 12 | 15 | 22 | 16 | 4 | 5 | 7 | 78 |
| 外部查询 | 16 | 22 | 28 | 22 | 3 | 4 | 6 | 88 |
| 内部逻辑文件 | 4 | 4 | 5 | 4 | 7 | 10 | 15 | 42 |
| 外部接口文件 | 2 | 2 | 3 | 2 | 5 | 7 | 10 | 15 |
| 未调整功能点总数 | | | | | | | | 320 |

$$P = \frac{a + 4m + b}{6}$$

P 估算值
 a 乐观值
 b 悲观值
 m 可能值

$$FP = r \times P$$

r 加权因子
 FP 总成本

Step2: 估计调整因子

- 由于软件规模会受到诸如性能、数据处理、可用性、可维护性等多种技术因素的影响，需要对计算得到的FP进行适当调整。
- 影响值：从0(不重要)~5(绝对必需)

| 技术因素 | 影响值 | 技术因素 | 影响值 |
|---------|-----|-----------|-----|
| 备份与恢复 | 4 | 主文件联机更新 | 3 |
| 数据通信 | 2 | 信息域值复杂度 | 5 |
| 分布式处理 | 0 | 内部处理复杂度 | 5 |
| 关键性能 | 4 | 设计可复用的代码 | 4 |
| 现有操作环境 | 3 | 设计中的转换与安装 | 3 |
| 联机数据输入 | 4 | 多次安装 | 5 |
| 多屏幕输入切换 | 5 | 易于变更的应用设计 | 5 |

$$FP_{estimated} = FP \times [0.65 + 0.01 \times \sum_{i=1}^{14} (F_i)] = 320 \times 1.17 = 375$$

Step3: 计算调整功能点和总成本

$$FP_{estimated} = FP \times [0.65 + 0.01 \times \sum_{i=1}^{14} (F_i)] = 320 \times 1.17 = 375$$

- 平均生产率(v): 6.5FP/pm-----统计值
- 月平均工资: 8000元
- 每个FP的成本(u): 1230元
- 总成本(C): $C = FP \times u = 375 \times 1230 = 461250$ 元
- 总工作量(PM): $PM = FP / v = 375 / 6.5 = 58$ 人月

基于过程的估算

- 将过程分解为一组较小的任务，并估算完成每个任务所需的工作量

- 沟通
- 策划:可行性分析/计划
- 建模:分析/设计
- 构建:编码/测试
- 部署
- 基于过程估算实例

| 活动 | 客户沟通 | 策划 | 风险分析 | 建模 | | 构建 | | 客户评估 | 合计 |
|-------------|------|------|------|------|------|------|------|------|------|
| | | | | 分析 | 设计 | 编码 | 测试 | | |
| 用户接口及控制设备 | | | | 0.5 | 2.5 | 0.4 | 5 | | 8.4 |
| 二维几何分析 | | | | 0.75 | 4 | 0.6 | 25 | | 7.35 |
| 三维几何分析 | | | | 0.5 | 4 | 1 | 3 | | 8.5 |
| 数据库管理 | | | | 0.5 | 3 | 1 | 1.5 | | 6 |
| ○ 计算机图形显示设备 | | | | 0.5 | 3 | 0.75 | 1.5 | | 5.75 |
| 外部设备控制功能 | | | | 0.25 | 2 | 0.5 | 1.5 | | 4.25 |
| 设计分析模块 | | | | 0.5 | 2 | 0.5 | 2 | | 5 |
| 合计 | 0.25 | 0.25 | 0.25 | 3.5 | 20.5 | 4.5 | 16.5 | | 46 |
| %工作量 | 1% | 1% | 1% | 8% | 45% | 10% | 36% | | |

月平均工资: 8000元
 总成本(c): $C=8000*46=368000$ 元

小结: 软件项目估算

- 除此之外, 还有其他很多估算方法;
- 不同的方法采用不同的计算公式, 考虑的因素不同, 复杂程度也不同;
- 都是根据实际项目的经验所总结出来的;
- 不能说“谁好谁坏”, 应用的时候可以依据自身的经验对其进行修正。
- 建议: 阅读有关COCOMOII的相关材料
 - COCOMO(ConstructiveCOstMOdel): 软件构造性成本模型
 - 主要用于工作量估算与成本估算
 - 是最广泛使用和最全面的软件估算模型

7 项目进度计划与监控

软件项目进度计划

- 目标:
 - 使软件项目在截止日期之前成功的完成并提交给客户;
- 活动:
 - Step1: 将项目划分为多个活动、任务。PBS、WBS

Step2: 确定任务项目依赖性。定义任务网络

Step3: 时间分配。为任务安排工作时间段

Step4: 确认任务资源。确认每个任务的资源需求

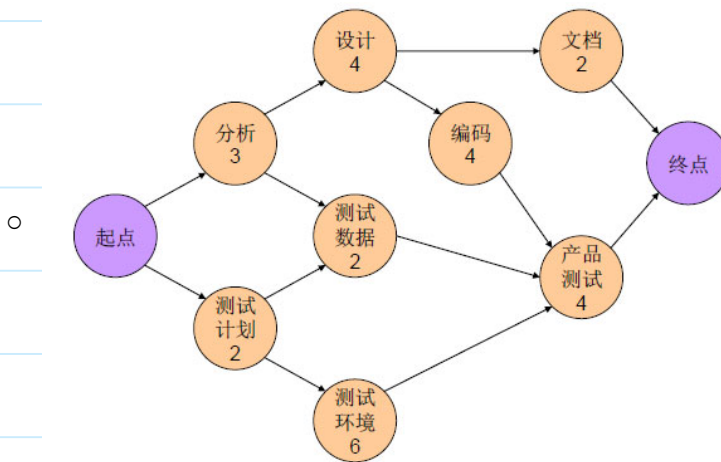
Step5: 确定责任。确定每个任务的负责人和参与人

Step6: 明确结果。明确任务的输出结果（文档或部分软件）

Step7: 确定里程碑(milestones)。确定任务与项目里程碑关联

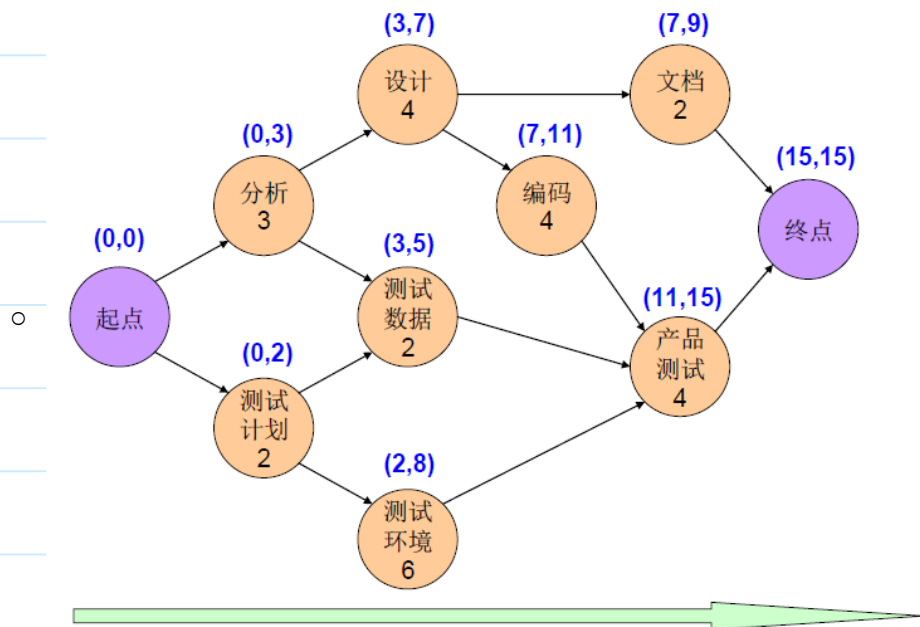
Step2: 定义任务网络

- 任务不是独立存在的，各任务在顺序上存在相互依赖关系。
- 项目管理里通常采用“网络图(NetworkDiagram)”的形式对此进行描述：

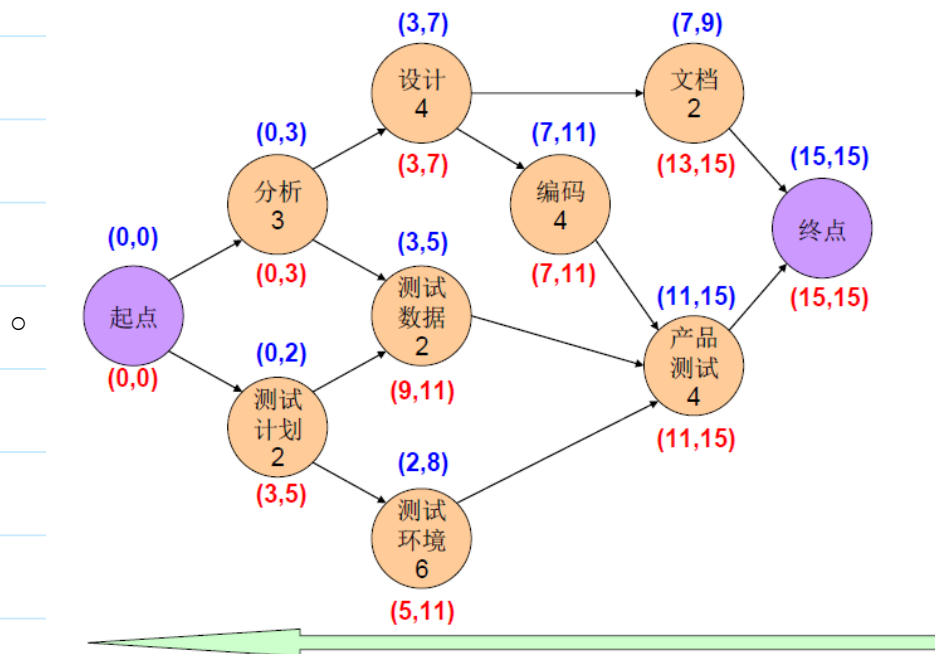


Step3: 时间分配

- 在绘制出任务网络图之后，下一步需要为每一个任务分配具体的执行时间。
- 两种具体的技术：
 - 程序评估及评审技术(PERT)
 - 关键路径方法(CPM)
- 步骤：
 - Step3.1: 标出任务的最早开始(ES)与结束时间(EF);
 - Step3.2: 标出任务的最晚开始(LS)与结束时间(LF);
 - Step3.3: 计算关键路径(CriticalPath)
 - Step3.4: 确定任务的开始/结束时间。
 - Step3.5: 绘制最终的任务进度安排(甘特图,GanttDiagram)
- Step3.1:标出最晚开始/结束日期



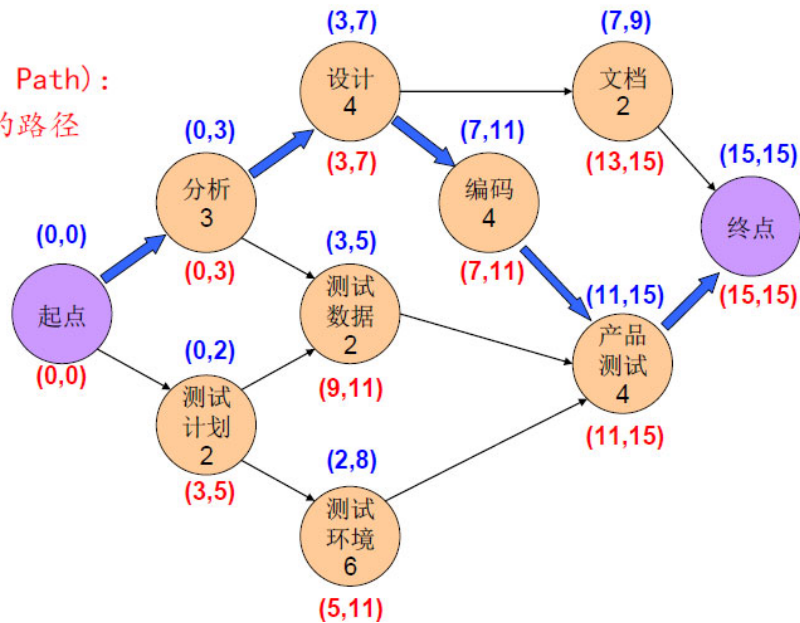
- Step3.2:标出最晚开始/结束日期



- Step3.3:计算关键路径(CriticalPath): 持续时间最长的路径

al Path):
关键的路径

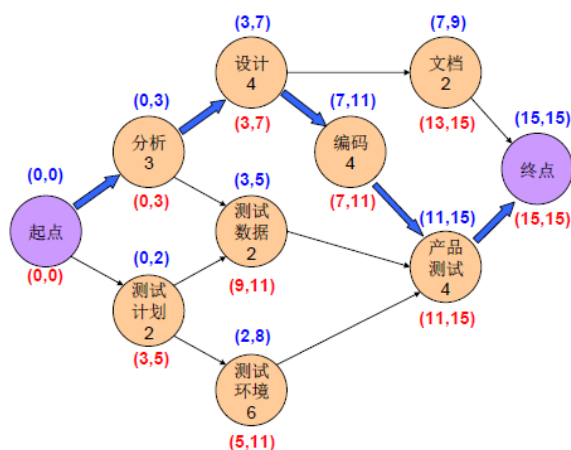
○



• Step3.4: 确定最终开始/结束日期

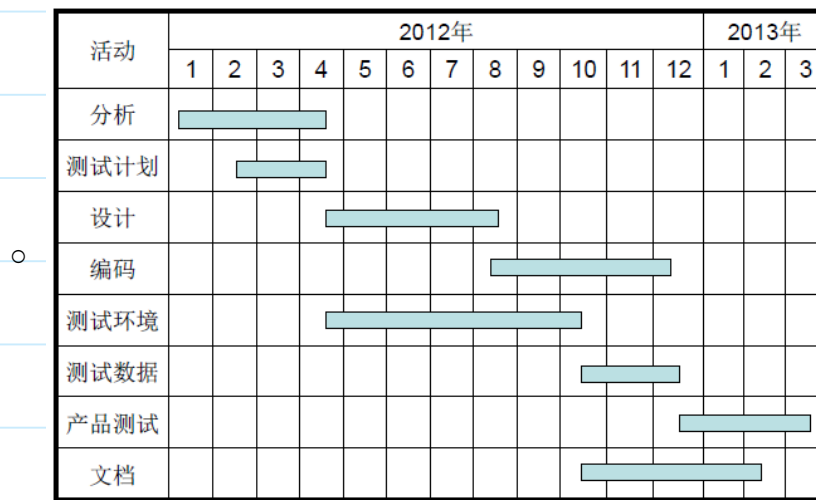
- [关键定义]松弛量(Slack): 在不影响最终交付日期的前提下, 一个任务最多可被推迟的总天数。
- 关键路径上的任务的松弛量=0;
- 先确定关键路径上各任务的日期;
- 然后确定其他任务的日期;
- 通过缩短关键路径上的任务的持续时间, 可极大优化整个项目的持续时间

○



• Step3.5: 绘制任务进度安排图

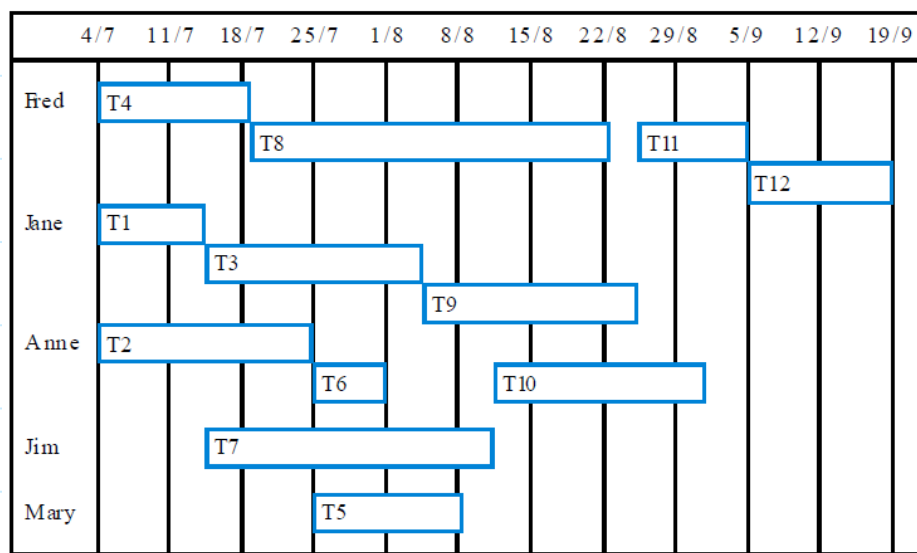
项目管理里通常采用甘特图(GanttChart)来描述任务的进度安排。



Step4~7:资源、产出与里程碑

- Step1: 将项目划分为多个活动、任务。PBS、WBS
- Step2: 确定任务项目依赖性。定义任务网络
- Step3: 时间分配。为任务安排工作时间段
- Step4: 确认任务资源。确认每个任务的资源需求工作量（人月）、资金、设备、环境等
- Step5: 确定责任。确定每个任务的负责人和参与人
- Step6: 明确结果。每一项任务的产出结果是什么？对应于PBS中的哪一部分？
- Step7: 确定里程碑(milestones)。项目的关键产出物，标志着某一阶段的完成。

人员/资源分配图



项目进度跟踪

项目进度表只是提供了一张进度路线图，在实际执行过程中，需要定期对其进行跟踪和控制，以决定是否需要对进度计划进行调整。

- 定期举行项目状态会议，各成员分别报告进展和存在问题；
- 评估软件工程过程中所进行的评审的结果；
- 判断项目里程碑是否在预定日期内完成；
- 比较项目的实际开始/结束日期与计划开始/结束日期；
- 与开发者非正式会谈，获取项目进展及可能出现问题的客观评估；
- 定量评估项目进展；
- 决策是否需要调整进度。

迭代开发项目进度跟踪

- 迭代项目问题：里程碑难以确认，为什么？
- 技术里程碑：OO分析完成
 - 已经定义和评审了所有的类、类层次
 - 已经定义和评审了与每一个类相关的属性和操作
 - 已经建立和评审了各类之间的关系
 - 已经建立和评审了行为模型
 - 已经确定了可复用的类
- 技术里程碑：OO设计完成
- 技术里程碑：OO测试完成

XP/Scrum敏捷开发中的进度计划与监控

- 以“迭代”为单位：每次迭代包含多少个用户故事或用例；
- 每次迭代为15-30天左右；
- 针对每个用户故事，团队成员联合估算和协商开发代价(时间)；
- 使用任务墙(TaskBoard)/燃尽图(BurndownChart)等作为进度监控工具，评估迭代的当前进展情况。

敏捷开发的项目管理工具：VersionOne

- <http://www.versionone.com>
- 敏捷领域最流行的商业化项目管理工具之一