复习-软件质量与管理

概述

- 软件危机
  - 落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象
- 软件工程
  - 是一门研究用工程化方法构建和维护有效的、实用的和高质量的软件的学科
  - 两大视角
    - 管理视角：能否复制成功
    - 技术视角：能都将问题解决得更好
- 软件项目管理
  - 管理的三大关键要素
    - 目标
    - 状态
    - 纠偏
  - 三大典型目标
    - 成本
    - 质量
    - 工期
  - 软件项目管理是应用方法、工具、技术以及人员能力来完成软件项目，实现项目目标的过程
- 软件过程
  - 狭义：软件活动及其顺序
  - 广义：狭义过程，技术，人员
    - 同义词：软件开发过程，软件开发方法
      - Cleanroom，XP，SCRUM，GATE，敏捷
      - **生命周期模型也算**：瀑布模型，迭代式模型，增量模型，螺旋模型，原型法
- 生命周期模型 与 软件过程的区别
  - 生命周期模型是对一个软件开发过程的人为划分
  - 生命周期模型是软件开发过程的主框架，是对软件开发过程的一种粗粒度划分
  - 生命周期模型往往不包括技术实践
- 软件过程管理
  - 管理的对象：软件过程
  - 管理的目的：为了让软件过程在开发效率、质量等方面有着更好性能绩效
  - 过程管理参考模型：CMM/CMMI，SPICE
  - 过程改进参考元模型：PDCA，IDEAL
- CMMI

附录2015标准答案，背那个吧

cmmi模型与敏捷方法是两个层面的东西，前者是过程改进参考模型，后者是项目管理方法，对立的基础不存在

  - 1：原始级别，开发相对混乱，主要依靠个人
  - 2：已经管理级别，项目小组级别体现出项目管理的特征，有项目计划、跟踪和需求管理等
  - 3：已经定义级别，管理活动和工程活动均已文档化、标准化，并集成到组织层面
  - 4：定量管理级别，采集详细的有关软件过程和产品质量的度量，可以进行有效预测
  - 5：持续优化级别，利用来自过程和来自新思想新技术的试验反馈信息，进行持续过程改进

> 软件过程管理是软件项目管理应该要实现目标（x）软件过程管理应有专人去做

> 在公司导入敏捷过程是我们今年过程改进的主要目标（x）敏捷不应该被作为改进的目标，他往往是最底层
>
> XP 与 CMM/CMMI 是对立的两种软件开发方法（x）CMMI 是过程管理，敏捷是软件过程，没有对立基础
>
> CMM/CMMI 不适合当今互联网环境的项目管理需求（x）CMMI 不是项目管理而是过程管理
>
> PDCA 和 IDEAL 不适合在敏捷环境中使用（x）他们适合任意环境下的过程管理需求
>
> 不同的软件开发过程应该使用不同的生命周期模型，反之亦如此（x）生命周期模型是人为划分的，没有什么绝对标准，主要看需要

软件过程历史

- 软件开发的本质难题
    - 可见性，复杂性，可变性，一致性
    - 三个本质难题因项目而异
    - 四大本质难题相互促进
    - 本质难题变化带动软件过程演变
- 软件发展三大阶段
    - 软硬件一体化（50s~70s）
        - 软件依附于硬件
            - 软件应用特征
                - 软件支持硬件完成计算任务
                - 功能单一
                - 复杂度有限
                - 几乎不需要需求变更
            - 软件开发特征
                - 硬件太贵
                - 团队以硬件工程师和数学家为主
            - 典型过程实践
                - 硬件开发流程
                - measure twice cut once
        - 软件作坊
            - 软件应用特征
                - 功能简单
                - 规模小
            - 软件开发特征
                - 很多非专业领域的人员涌入软件开发领域
                - 高级程序语言出现
                - 质疑权威文化盛行
            - 典型过程实践
                - code and fix
    - 软件成为独立产品
        - 软件应用特征
            - 摆脱了硬件束缚（OS）
            - 功能强大
            - 规模和复杂度剧增
            - 个人电脑出现，普通人成为软件用户（需求多变，兼容性要求）

        - 来自市场的压力
    - 典型过程实践
        - 形式化方法（在扩展性和可用性方面存在不足）
        - 结构化程序设计和瀑布模型（重文档，慢节奏）
        - 成熟度运动
- 网络化和服务化
    - 软件应用特征
        - 功能更复杂，规模更大
        - 用户数量急剧增加
        - 快速演化和需求不确定
        - 分发方式的变化（SaaS）
    - 典型过程实践
        - 迭代式
        - 敏捷运动
            - **敏捷宣言**
                - 个体和互动胜过流程和工具
                - 可以工作的软件胜过详尽的文档
                - 客户合作胜过合同谈判
                - 响应变化胜过遵循计划
                - 尽管右项有其价值，我们更重视左项的价值
        - XP，Scrum
        - Kanban
            - 可视化工作流，限定 WIP，管理周期时间
        - 开源
            - 一种基于并行开发模式的软件开发的组织与管理方式
            - 需要足够多的测试者和合作开发者

我们应该如何正确理解瀑布模型
1. 瀑布模型不是单一模型，是一系列模型，覆盖最简单场景（过程元素少）到最复杂的场景（过程元素多）
2. 软件项目应该更加实际情况选择合适过程元素的瀑布模型，基本原则是，项目面临困难和挑战越多，选择的模型应该越复杂；
3. 软件项目团队往往低估项目的挑战，选择了过于简单的瀑布模型

团队动力学

- 自主团队
    - 必要性
        - 软件开发是一项既复杂又富有创造性的知识工作
        - 软件开发——智力劳动
            - 处理和讨论极其抽象的概念
            - 把不同的部分（不可见）整合成一个可以工作的系统

补充：全身心地参与，努力做出卓越的工作

    - 应该具备的特征
        - 自行定义项目的目标
        - 自行决定团队组成形式以及成员的角色
        - 自行决定项目的开发策略
        - 自行定义项目的开发过程
        - 自行制定项目的开发计划
        - 自行度量、管理和控制项目工作
- 知识工作
    - 管理知识工作的关键规则：管理者无法管理工作者，知识工作者必须实现并且学会自我管理
    - 要自我管理，知识工作者必须
        - 有积极性

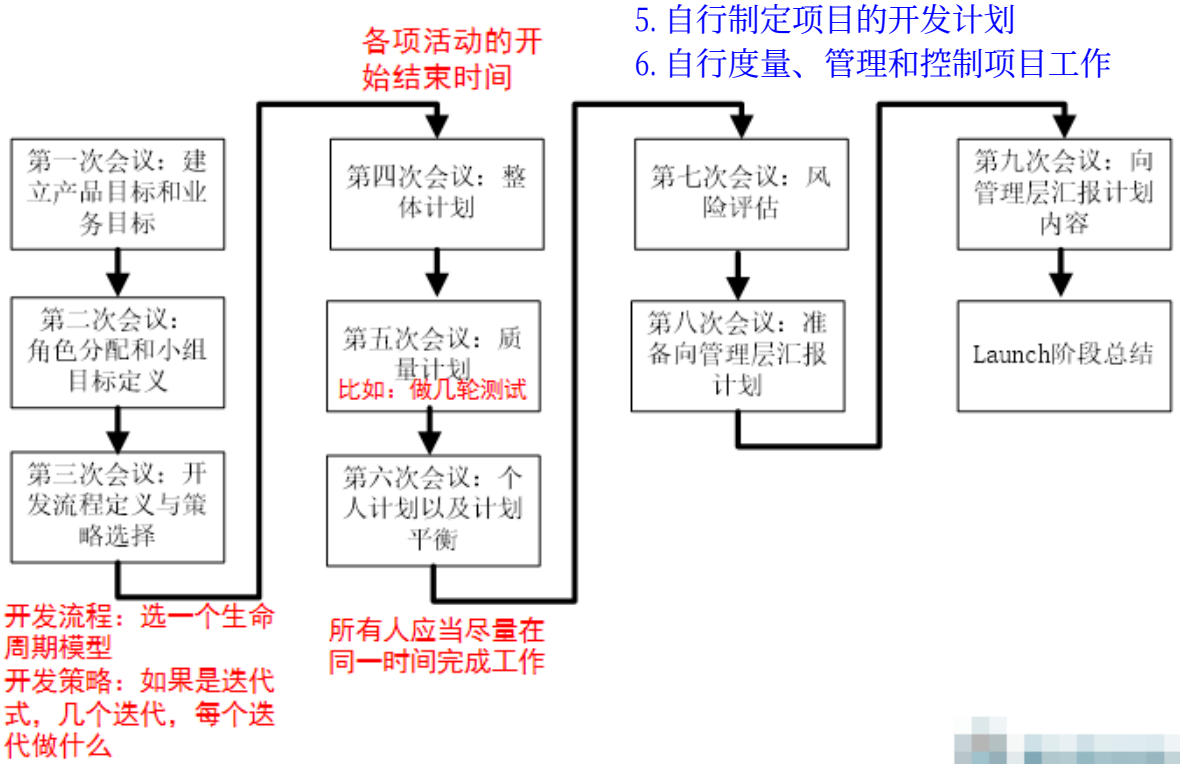
        - 能做出准确的估算和计划
        - 懂得协商承诺
        - 有效跟踪他们的计划
        - 持续地按计划交付高质量产物
    - 领导者特点
        - 诚实
        - 能力强
        - 洞察力
        - 鼓动力
- 马斯洛需求层次理论
    - Lv.1 生理需求
    - Lv.2 安全感
    - Lv.3 爱和归属感
    - Lv.4 获得尊敬
    - Lv.5 自我实现
    - 低层次的需求必须在高层次需求满足之前得到满足
    - 满足高层次的需求的途径比满足低层次的途径更为广泛
- 激励手段
    - 威逼（Lv.2）（交易型）
    - 利诱（Lv.1-2）（交易型）
    - 鼓励承诺（Lv.4）（转变型）
- 期望理论
    - M = V * E
        - M：成果
        - V：相信努力能产生成功的结果
        - E：相信自己会因为成功得到回报

框中是马斯洛的人的需求层次有哪几个，这样的分层对软件开发有什么启发

TSP

自主团队应该具备的特征：
1. 自行定义项目的目标
2. 自行决定团队组成形式以及成员的角色
3. 自行决定项目的开发策略
4. 自行定义项目的开发过程
5. 自行制定项目的开发计划
6. 自行度量、管理和控制项目工作

- TSP 会议流程



- TSP 角色

- 项目组长
  - 工作内容
    - 激励团队成员努力工作
    - 主持项目周例会
    - 每周汇报项目状态
    - 分配工作任务
    - 维护项目资料
    - 组织项目总结
- 计划经理
  - 工作内容
    - 带领项目小组开发项目计划
    - 带领项目小组平衡计划
    - 跟踪项目进度
    - 参与项目总结
- 开发经理
  - 工作内容
    - 带领团队制定开发策略（会议 3）
    - 带领团队开展产品规模估算和所需时间资源的估算
    - 带领团队开发需求规格说明
    - 带领团队开发高层设计
    - 带领团队开发设计规格说明
    - 带领团队实现软件产品
    - 带领团队开展集成测试和系统测试
    - 带领团队开发用户支持文档
    - 参与项目总结
- 质量经理
  - 工作内容
    - 带领团队开发和跟踪质量计划
    - 向项目组长警示质量问题
    - 软件产品提交配置管理之前，对其进行评审，以消除质量问题
    - 项目小组评审的组织者和协调者
    - 参与项目总结
- 过程经理
  - 工作内容
    - 带领团队定义和记录开发过程并且支持过程改进
    - 建立和维护团队的开发标准（编码规范，模块命名等）
    - 记录和维护项目的会议记录
    - 参与项目总结
- 支持经理
  - 工作内容
    - 带领团队识别开发过程中所需要的各类工具和设施
    - 主持配置管理委员会，管理配置管理系统
    - 维护软件项目的词汇表
    - 维护项目风险和问题跟踪系统
    - 支持软件开发过程中复用策略的应用
    - 参与项目总结

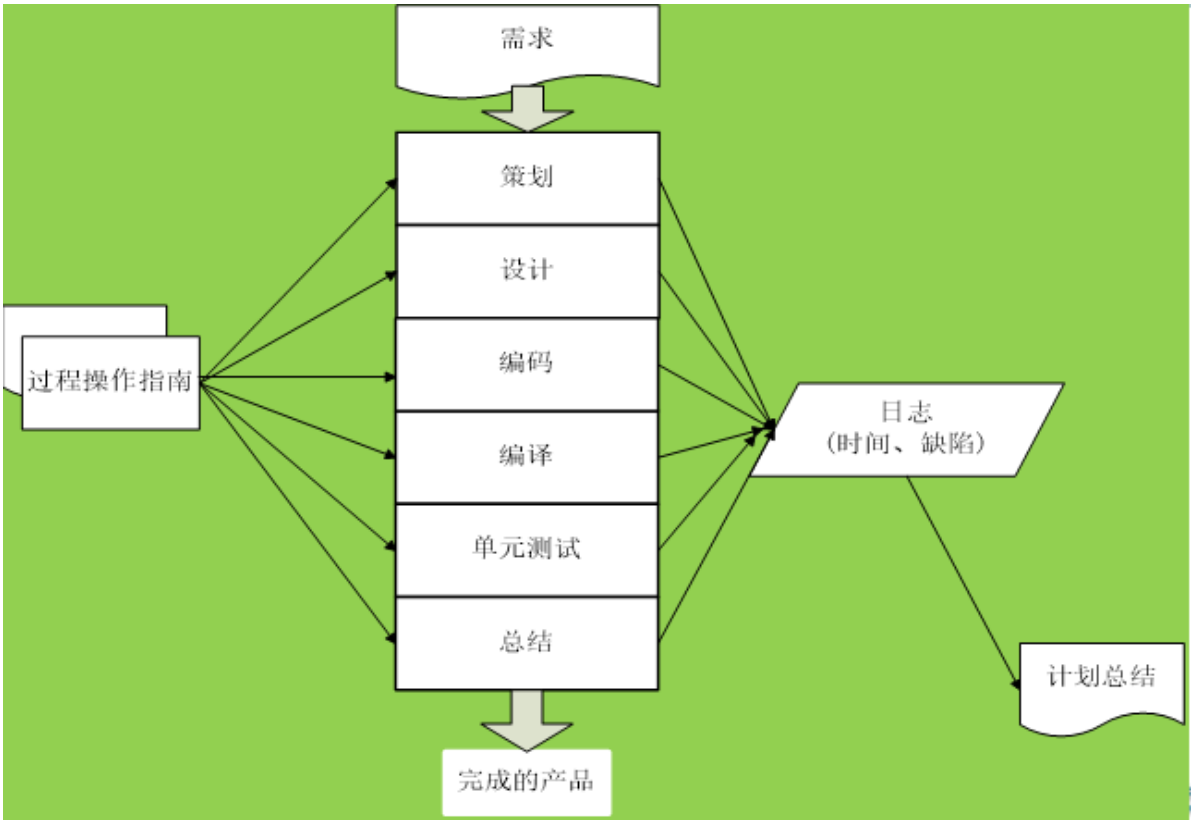
Scrum

- 产品负责人
    - 职责是将开发团队开发的产品价值最大化
    - 是负责管理产品待办列表的唯一负责人
        - 清晰地表述产品待办列表项
        - 对产品待办列表项进行排序
        - 优化开发团队所执行工作的价值
        - 确保产品待办列表对所有人是透明和清晰的，同时显示 Scrum 团队下一步要做的工作
        - 确保开发团队对产品待办列表项有足够深的了解
- 开发团队
    - 是自组织的，自行决定如何把产品待办列表变成潜在可发布的功能增量
    - 是跨职能的，团队作为一个整体，拥有创建产品增量所需的全部技能
    - 不认可开发团队成员的任何头衔，不管其承担何种工作都叫开发人员
    - 不认可开发团队中所谓的“子团队”
    - 每个成员也许有特长和专注的领域，但是责任属于整个开发团队
- Scrum Master
    - 促进和支持 Scrum
    - 帮助每个人理解 Scrum 理论、实践、规则和价值
    - 是一位服务型领导，负责内外沟通
    - 服务于产品负责人
        - 帮助 Scrum 团队理解为何需要清晰且简明的产品待办列表项
        - 找到有效管理产品待办列表的技巧
        - 确保产品负责人懂得如何来安排产品待办列表使其达到最大化价值
        - 当被请求或需要时，引导 Scrum 事件
    - 服务于开发团队
        - 作为教练在自组织和跨职能方面给予开发团队以指导
        - 帮助开发团队创造高价值的产品
        - 移除开发团队工作进展中的障碍
        - 当被请求或需要时，引导 Scrum 事件
    - 服务于组织
        - 带领并作为教练指导组织采纳 Scrum
        - 在组织范围内规划 Scrum 的实施
        - 帮助员工和利益攸关者理解并实施 Scrum 和经验导向的产品开发
        - 引发能够提升 Scrum 团队生产率的改变
        - 与其他 Scrum Master 一起工作，增强组织中 Scrum 应用的有效性

估算，计划和跟踪

PSP

- 什么是 PSP
    - PSP 是包括了数据记录表格、过程操作指南和规程在内的结构化框架
- 典型 PSP 过程



- PSP 基本度量项
  - 规模：规模是连接下面三个的桥梁，没有规模数据，这些数据都不能用
  - 时间
  - 缺陷
  - 日程
- 时间日志
  - 序号
  - 所属阶段
  - 开始时间
  - 结束时间
  - 中断时间
  - 净时间
  - 备注信息
- 缺陷日志
  - 序号
  - 发现日期
  - 注入阶段
  - 消除阶段
  - 消除时间
  - 关联缺陷
  - 简要描述
- PROBE 估算
  - 原理
    - 设立合理的代理作为精确度量和早期规划需要的度量之间的桥梁
    - 使用相对大小，而非绝对大小
  - 流程

谈谈你对项目估算（时间和规模）的认识（包括原理、方法和目标等），并简要解释应用PROBE 方法进行估算的优缺点。

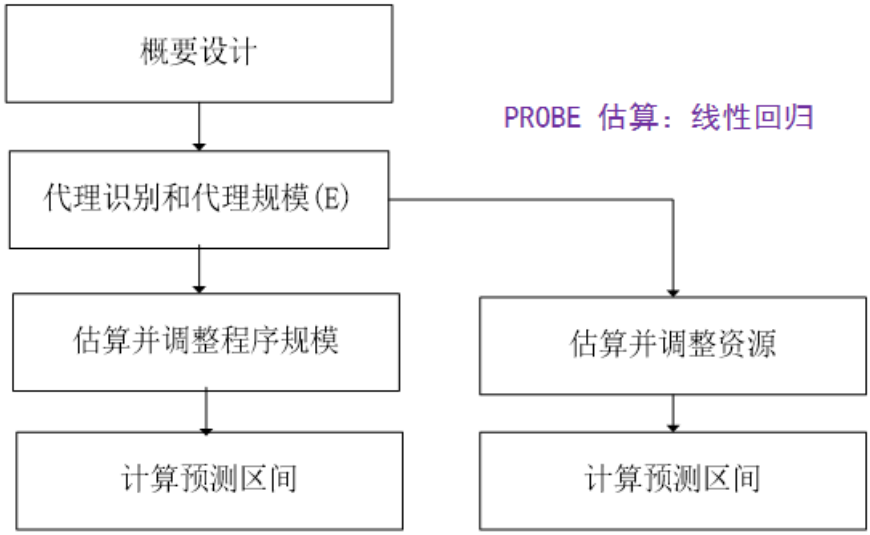
1 规模估算往往可以依据历史数据来完成，其原因在于规模估算结果的偏差产生原因相对客观，偏差可以用以修正新的估算结果。

2 时间估算的偏差产生原因更加复杂，一方面和规模有关，另外一方面，跟人的主观能动性有关，因此，时间估算偏差的原因可能估算结果本身，这使得历史数据中时间偏差可参考价值不大。

3 从上述讨论可以得出，对于估算来说，本质上是一种猜测，追求的目标应该是一致性以及估算结果的使用者对估算结果的信心。

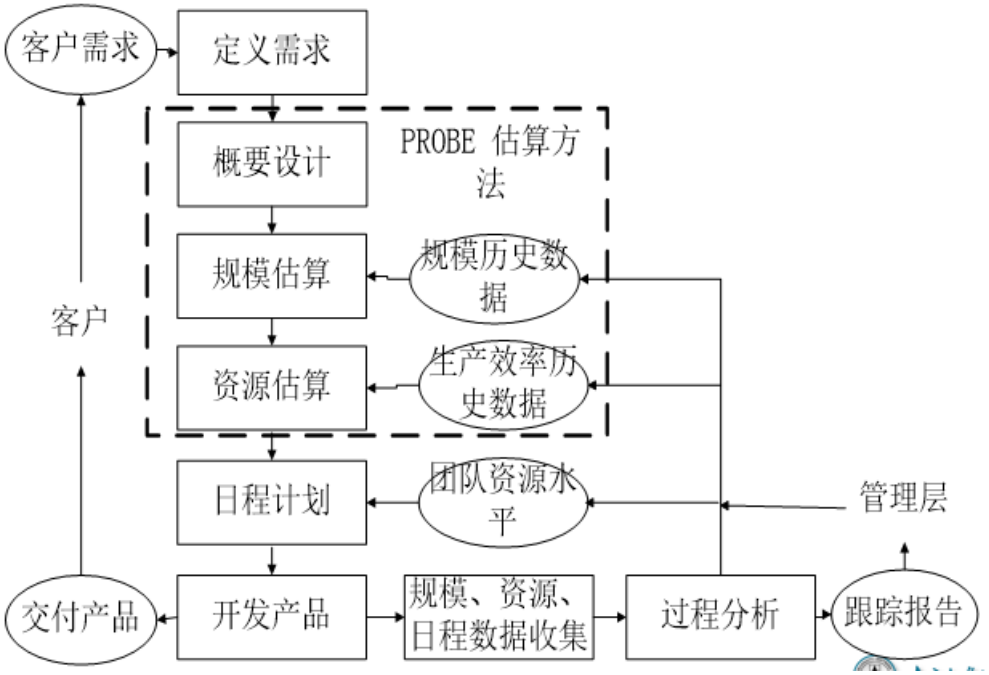
4 PROBE方法通过定义的估算过程和数据收集以及使用的框架，使得估算结果可以尽可能一致，从而使得一些统计方法可以用来调整估算结果，增强用户对估算结果的信心。

5 但是这种估算方法非常依赖高质量的历史数据，一旦数据不完整或者缺失，就可能导致估算结果有显著偏差



PROBE 估算：线性回归

- 通用计划框架



- 规模估算要求

PROBE方法	数据要求	数据质量要求	计算方法
A 了多次 B 方 ，当积累的数 能够形成相对 小矩阵，才能 用 A 方法	3组或者3组以上代理规模(E)与实际程序规模。	r≥0.7； s≤0.05； β0≤估算结果的25%； 0.5≤β1≤2；	略。
B	3组或者3组以上计划程序规模与实际程序规模。	r≥0.7； s≤0.05； β0≤估算结果的25%； 0.5≤β1≤2；	略。
C	有历史数据	无	按比例调整。
D	没有历史数据	无	猜测。

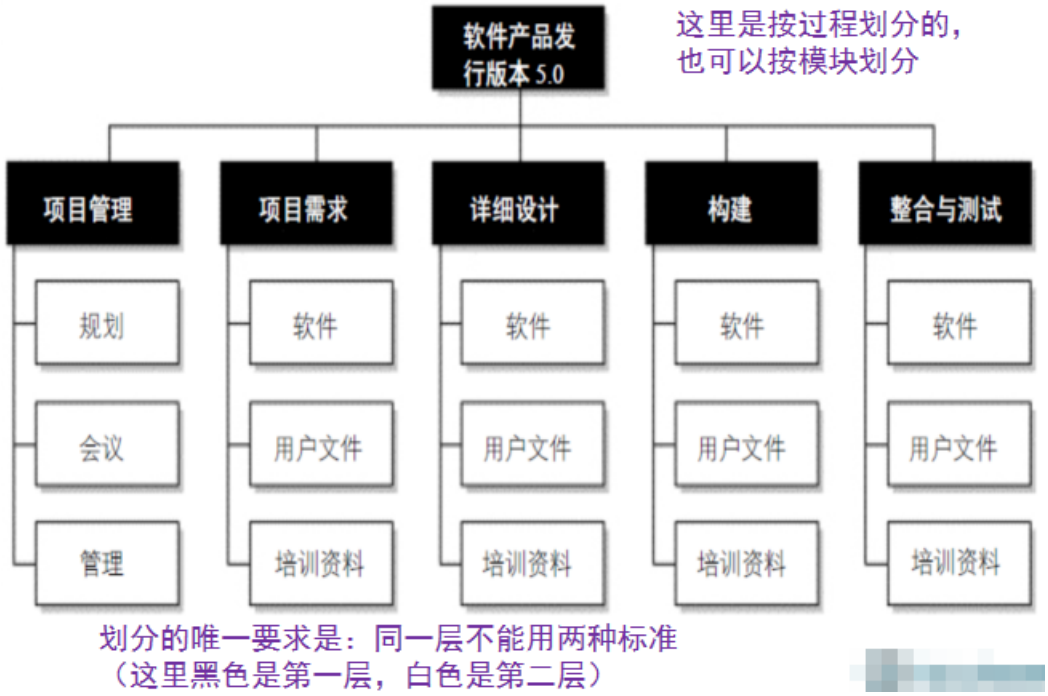
- 时间估算要求

PROBE 方法	数据要求	数据质量要求	计算方法
A	3组或者3组以上代理规模(E)与实际开发时间。	r≥0.7； s≤0.05； β0显著小于估算结果； β1≤0.5×（历史生产效率的倒数）	略。
B	3组或者3组以上计划程序规模与实际开发时间。	r≥0.7； s≤0.05； β0显著小于估算结果； β1≤0.5×（历史生产效率的倒数）	略。
C	有历史数据	无	按比例调整。
D	没有历史数据	无	猜测。

probe a方法估算时间，不用历史数据中的生产效率数据原因：在估算资源需求的时候，生产效率一般在分母上，考虑到个体软件工程师生产效率波动，易导致估算偏差范围变大

WBS

- 示例

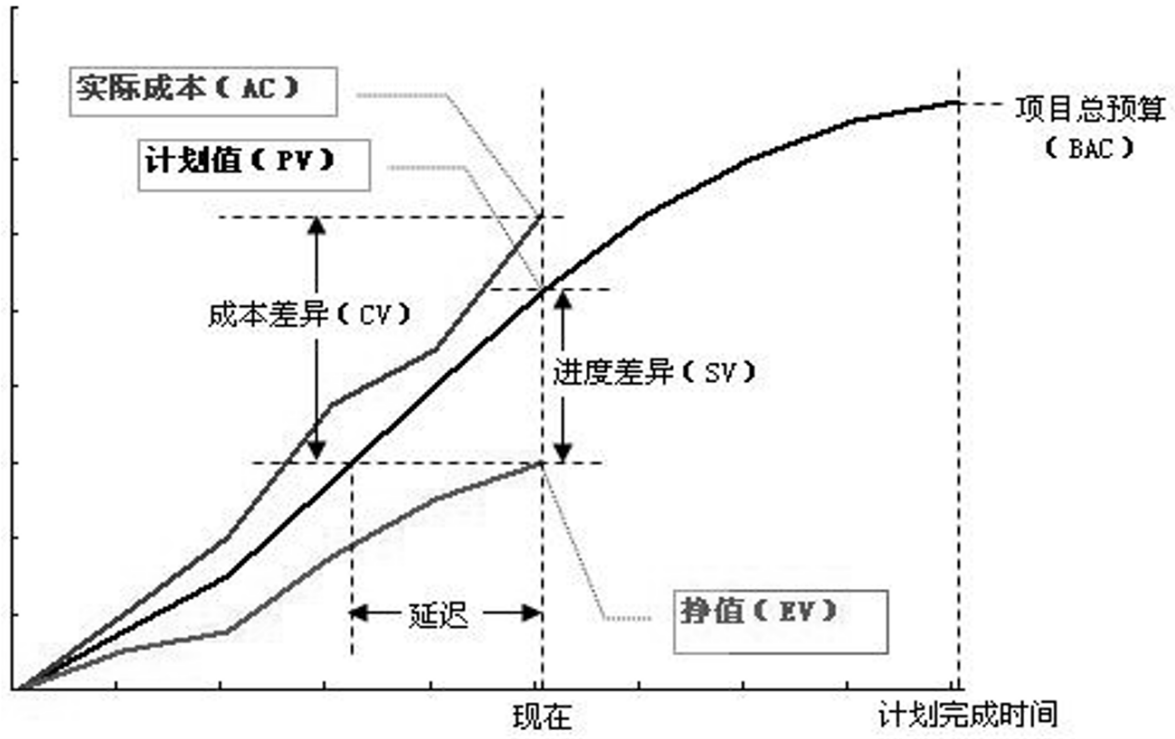


- 好的 WBS 检查标准
  - 最底层要素不能重复
  - 所有要素必须清晰完整定义
  - 最底层要素必须有定义清晰的责任人
  - 最底层的要素是实现目标的成分必要条件
- 范围管理
  - 确保项目做且只做成功完成项目所需的全部工作
  - WBS 为范围管理提供了基准
    - 收集需求
    - 定义范围
    - 创建 WBS
    - 核实范围
    - 控制范围变更

- 开发策略
  - 产品组件开发顺序
  - 产品组件获得方式：新开发？第三方？改写原有？
- 日程计划
  - 典型计划流程
    - 估算规模
    - 估算资源
    - 规划日程
  - 所需信息
    - 任务清单
    - 任务顺序
    - 人员资源水平
- 质量计划
- 风险计划
  - 风险类别
    - 可能性
    - 影响程度

挣值管理方法

- EVM
- 是用来客观度量项目进度的一种项目管理方法
  - 每项任务实现附以一定价值
  - 完成该项任务，就获得相应价值
- 分析图



- 局限性
  - 一般不能应用软件项目的质量管理
  - 需要定量化的管理机制，这就使其在一些探索型项目以及常用的敏捷开发方法中的应用受到限制

    - 完全依赖项目的准确估算，然而在项目早期，很难对项目进行非常准确的估算
- 其它计划跟踪
    - 里程碑评审
        - 项目相关的承诺，如日期、规格、质量等
        - 项目各项计划的执行状况
        - 项目当前的状态讨论
        - 项目面临的风险讨论

质量管理

质量

- 质量的概念
    - 与软件产品满足规定的和隐含的需求能力有关的特征或者特性的全体
    - 分为内外两部分
        - 外部特性：面向软件产品的最终用户
        - 内部特性：面向开发人员
- PSP 质量策略
    - 用 缺陷管理 代替 质量管理
    - 高质量产品 自然意味着 组成软件产品的各个组件基本无缺陷
    - 各个组件的高质量是通过高质量评审来实现的
- 缺陷消除方式（消除缺陷所需的平均时间 / min）
    - 设计评审：5
    - 设计检查：22
    - 代码评审 ： 2
    - 代码检查 ： 25
    - 单元测试 ： 32
    - 系统测试 ： 1405
- 测试消除缺陷的典型流程
    - 发现待测程序的一个异常行为
    - 理解程序的工作方式
    - 调试程序，找出出错的位置，确定出错原因（最耗时）
    - 确定修改方案，修改缺陷
    - 回归测试，以确认修改有效
- 评审发现缺陷的典型流程
    - 遵循评审者的逻辑来理解程序流程
    - 发现缺陷的同时，也知道了缺陷的位置和原因
    - 修正缺陷
- 为什么要通过评审保证质量?
    - 答：经济
- PSP 质量评审
    - 评审检查表
        - 以前讲过的什么有助于形成个性化的审查表?
            - 答：缺陷日志（注入，消除，根本原因）
    - 质量控制指标
        - Yield
            - 用以度量每个阶段在消除缺陷方面的效率

        - Phase Yield = 某阶段发现缺陷数 / (某阶段注入缺陷数)
        - Process Yield = 第一次编译前发现的缺陷数 / 第一次编译前注入的缺陷数
        - 大致过程：阶段一注入 => Phase Yield => 阶段二注入 => Phase Yield（整体是 Process Yield）

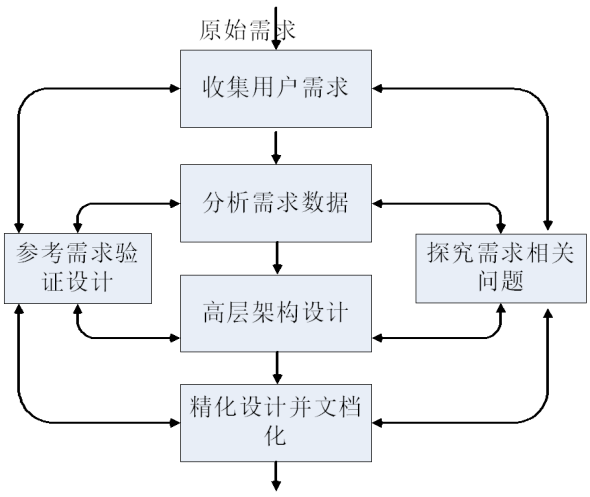
	注入缺陷数	**消除缺陷数**	**遗留缺陷数**	**phase yield**	**process yield**
DLD（详细设计）	10 (+4)	0	10	0	
DLDR（详细设计审查）	0	6	4	60%	
CODE（编码）	20 (+8)	0	24	0	
CR（代码审查）	0	12	12	50%	
UT（单元测试）	0	12	0 (+12)	100%	(=>50%)

        - 然而，遗留缺陷数并不是一个客观事实！！！
        - 所以只能估算，不能度量
            - 假设最后一步干掉一半错误，强行增加 12 个缺陷，保证 process yield 至少 50%
            - 按比例把这 12 个缺陷分配到注入阶段
- A/FR
    - 质检失效比
    - A/FR = PSP 质检成本 / PSP 失效成本
    - PSP 质检成本 = 设计评审时间 + 代码评审时间
    - PSP 失效成本 = 编译时间 + 单元测试时间
    - 理论上，A/FR 越高质量越高
    - 过高说明做了过多评审，反而可能导致开发效率下降
    - 一般 2.0 就够了
- PQI
    - 过程质量指标
    - 5 个数据的乘积
        - 设计质量：min { 设计时间 / 编码时间 , 1 }
        - 设计评审质量：min { 2 * 设计评审时间 / 设计时间 , 1 }
        - 代码评审质量：min { 2 * 代码评审时间 / 编码时间 , 1 }
        - 代码质量：min { 10 / 代码编译缺陷密度 , 1 }
        - 程序质量：min { 10 / (代码单元测试缺陷密度 + 5) , 1 }
    - 作用
        - 了解模块的开发质量
        - 帮我们做质量规划
        - 为过程改进提供依据（五边形图，哪个低就加长哪个的时间）
- Review Rate
    - 评审速度
    - 高质量的评审需要软件工程师投入足够的时间进行评审（意思速度要慢？？）
    - 代码评审速度 < 200代码行/小时
    - 文档评审速度 < 4页/小时
- DRL

        - 缺陷消除效率比
    - 度量不同缺陷消除手段的效率
        - 以某个测试阶段（一般为单元测试）每小时发现的缺陷数为分母，直接取比值
            - 希望其大于等于 1
    - 打印后评审效果更好
        - 单个屏幕可以展现的内容比较有限
        - 打印之后，评审人员完全脱离计算机环境，更容易集中注意力
- 先编译（单元测试）or 先评审？
    - 当我们的目的是 “找出所有错误” 时，评审时间与代码是否编译无关
    - 先评审可能发现语法错误，进而减少编译的时间

设计

- 设计与质量的关系
- 低劣的设计是导致在软件开发中返工、不易维护以及用户不满的主要原因
    - 充分设计可以显著减少最终程序的规模，提升质量
    - 设计本身也是一种排错的过程
- PSP 设计过程



最终设计结果

- 设计的内容

	动态信息	静态信息
外部信息	交互信息（服务，消息）	功能（类结构）
内部信息	行为信息（状态机）	结构信息（属性，业务逻辑）

- 设计模板
    - OST
        - 操作规格模板（外部 - 动态）
        - 描述用户与待设计系统的正常和异常情况下的交互
        - 可以用来定义测试场景和测试用例，也可以作为和系统用户讨论需求的基础
        - 用例图，时序图
    - FST
        - 功能规格模板（外部 - 静态）

        - 描述是系统对外的接口
        - 提供的典型信息包括类和继承关系，外部可见的属性和外部可见的方法等
        - 消除二义性非常重要，尽可能用形式化符号来描述方法等行为
        - 类图相似，但没有描述
    - SST
        - 状态规格模板（内部 - 动态）
        - 精确定义程序的所有的状态、状态之间的转换以及伴随着每次状态转换的动作
        - 状态图相似，但没有状态转换条件、状态转换中的动作对应的图示
    - LST
        - 逻辑规格模板（内部 - 静态）
        - 精确描述系统的内部静态逻辑
        - 消除二义性非常重要，建议用伪代码配合形式化符号来描述设计结果
        - UML 中没有对应的图示
- 设计层次
    - 系统需求定义
    - 系统规格说明
    - 系统高层设计
        - 子系统规格说明
        - 子系统高层设计
            - 组件规格说明
            - 组件高层设计
                - 模块规格说明
                - 模块详细设计
- 设计验证
    - 状态机验证
        - 概念
            - 完整
            - 正交
        - 验证方法
            - 消除死循环（绕不出去）和陷阱状态（到不了结束状态）
            - 验证完整性和正交性
            - 评价状态机是否正确体现设计意图
        - 真值表

n	< nMax				= nMax				> nMax			
Pass word	Valid		!Valid		Valid		!Valid		Valid		!Valid	
ID	Valid	!Valid	Valid	!Valid	Valid	!Valid	Valid	!Valid	Valid	!Valid	Valid	!Valid
Next for Each Defined Transition Condition												
A		CID	CID	CID								
B	End				End				End			
C					End	End	End	End	End	End	End	End
Resulting Values of Fail for Each Defined Transition Condition												
A												
B	F				F				F			
C					T	T	T	T	T	T	T	T

- 符号化执行验证
  - 步骤
    - 识别伪码程序中的关键变量
    - 将这些变量用代数符号表示，重写伪码程序
    - 分析伪码程序的行为
  - 验证 X:=X+Y Y:=X-Y X:=X-Y

指令	X	Y
初始化	A	B
X:=X+Y	A+B	
Y:=X-Y		A
X:=X-Y	B	
结果	B	A

  - 优缺点
    - 实施简单
    - 通常用在验证一些复杂算法中，特别是对遗留系统的改造中，往往应用这种方法来识别和理解原有的设计
    - 但不适用于有复杂逻辑的场合，纯手工的验证方法也容易引入一些人为的错误
- 执行表验证：就一页，不重要（看上去和上面的表格没什么区别）
- 跟踪表验证：就一页，不重要
- 正确性验证
  - 步骤
    - 分析和识别用例
    - 对于复杂伪码程序的结构，应用正确性检验的标准问题逐项加以验证
    - 对于不能明确判断的复杂程序结构，使用跟踪表等辅助验证
  - 比如：while-do 正确性检验
    - condition 是否最终一定会为 false，从而使得循环可以结束
    - condition 为 true 时，其与 循环前执行一次循环体再加一个循环结构 的执行结果是否一致
    - condition 为 false 时，循环体内所有变量是否未被修改

团队工程开发

需求开发

- 需求是一切工程活动的基础
- 需求分类
  - 客户需求：一个问题，可以提出各种限制，比如预算等
  - 产品需求：这个问题的解决方案
- 需求获取
  - 尽可能识别客户的期望与所受的限制
  - 客户在描述其期望时一般不会显式提出这些额外的需求
- 需求汇总
  - 整理各种来源的信息，识别缺失的信息

    - 解决冲突的需求
    - 总结：将客户需求转化为产品需求
- 需求验证
    - 对需求进行分析和确认，以确保符合使用者预期
    - 典型活动
        - 建立和维护操作概念和相关的场景
        - 分析需求
        - 确认需求（不是简单的签字而已）
- 需求文档制作
    - 需求开发工作完成的一个基本标志是形成了一份完整的、规范的、经过评审的需求规格说明书
    - 需求规格说明书的编制是为了使用户和软件开发者对软件的初始规定有一个共同的理解，使之成为整个开发工作的基础
- 优秀的需求规格文档特征
    - 内聚：仅仅用以说明一件事情
    - 完整：不能遗漏信息
    - 一致：各个条目和章节不能互相矛盾，与外部参考资料也不能
    - 原子：尽可能避免连接词，如果需要描述多项内容，可以分别用简单语句加以描述
    - 可跟踪：客户需求、产品需求以及产品组件需求必须可以双向跟踪
    - 非过期：内容必须体现对于项目的最新认识
    - 可行性：内容应该在项目所拥有的资源范围内可以实现
    - 非二义性：描述应当尽可能清晰、客观，不能有含糊不清或多种理解的情形
    - 强制：可选的需求内容要么不要出现，要么以明确的方式标注
    - 可验证：描述应当便于在后期开发过程中进行验证

团队设计

- 设计过程基本同 PSP
- 团体智慧
    - 确定整体架构之前很难进行分工？
        - 视软件系统的规模而定，选择适当人数的团队成员参与整体架构的开发，而其他人员参与架构的评价和关键技术问题的验证
    - 怎样提高团队成员在讨论和评审会议中的参与程度？
        - 会议协调者采取适当方法（这不等于没说吗？？？）
- 设计标准
    - 命名规范：系统，子系统，组件，模块，程序，文件，变量，参数……
    - 接口标准：接口格式，设计原则（如高内聚、低耦合）
    - 系统出错信息
    - 设计表示标准：设计工作产物应当满足的标准（PSP 设计模板可以作为基础）
- 复用性支持
    - 在设计阶段必须要充分考虑复用的可能
    - 复用管理流程
        - 复用接口标准
        - 复用文档标准：对于可复用组件必须提供详细支持文档，以明确组件功能、调用方式、返回值类型以及可能的异常信息
        - 复用质量保证机制：进行充分的测试，根据过程数据来判断复用组件的质量
- 可测试性考虑
    - 可能减少测试代码的数量：主要通过合理的架构设计来体现
    - 制作合理的测试计划：完整的设计工作和操作场景定义有帮助

- 可用性考虑
    - 可用性的问题应当在设计阶段就开始考虑，而不能推延到实现阶段
    - 针对每一个关键功能都定义操作概念和操作场景
    - 分析操作场景以确保软件系统开发完成之后，使用者会满意
    - 必要时，可以邀请最终用户参与场景的评审，使用原型等技术，更好的把握用户真实意图

实现策略

- 评审
    - 在设计中，自顶向下、逐层精化
    - 在实现中，应当更多考虑是否便于评审，因此，建议自底向上
    - 先实现底层的内容，对其评审保证质量，再进行高层实现
- 复用
    - 上面说到的自底向上
    - 编码注释
    - 站立式会议
- 可测试性
    - 实现的计划必须与测试计划一致，以免集成测试时有模块没有实现

集成策略

- 大爆炸
    - 将所有已完成的组件放在一起，进行一次集成
    - 优点
        - 最快
        - 需要测试用例最少
    - 缺点
        - 需要组件质量很高，否则难以定位缺陷
        - 系统越复杂，规模越大，问题越突出
- 逐一添加
    - 与大爆炸相反，一次只添加一个组件
    - 优点
        - 容易定位缺陷
        - 在组件质量不高的情况下，每次集成都有坚实的质量基础
    - 缺点
        - 需要测试用例最多
        - 耗时
- 集簇
    - 选择部分组件进行集成，然后以集成后的组件继续进行较高层次的集成
        - 功能相似或有关联的模块优先进行集成，形成可工作的组件
    - 优点
        - 可以尽早获得一些可以工作的组件，有利于其他组件测试工作的开展
    - 缺点
        - 过于关注个别组件，不能尽早发现系统层面的缺陷
- 扁平化
    - 优先集成高层的部件，然后逐步将各个组件、模块的实现加入系统
    - 优点

        - 可以尽早发现系统层面的缺陷
    - 缺点
        - 需要大量的 "桩"，这种方式往往不能覆盖整个系统应该处理的多种状态

验证与确认

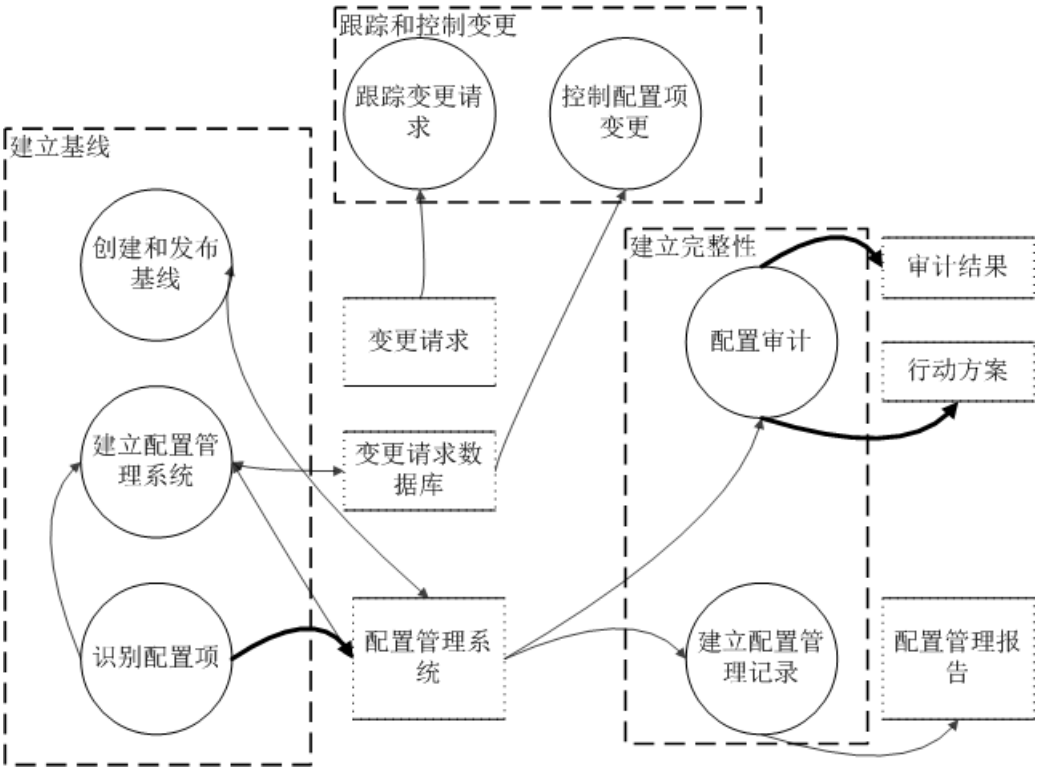
- 如何区分?
    - 验证（Verification）
        - 保证解决方案被正确地实现（解决方案有没有用不关我事）
    - 确认（Validation）
        - 确认客户的问题被你解决了（可能实现错了，但是歪打正着满足了客户需求）
- 分类
    - 验证：单元测试
    - 确认：需求评审；验收测试
    - 简单理解：开发过程的一头一尾（需求和验收）是确认，中间工作都是验证
    - 正确理解：两个活动交织贯穿整个开发过程
- 流程
    - 环境准备
        - 同行评审：准备文件材料、人员、场所
        - 测试：模拟器，场景生成程序，环境控制接口
        - 确认：模拟真实环境，或明确模拟环境与真实环境的差别
    - 对象选择
        - 并不是所有的都要，按照相应的计划
        - 产品：面向客户；工作产品：过程的直接结果
        - 确认：产品；验证：工作产品
    - 活动实施
        - 主要就是评审和测试
    - 结果分析
        - 找出潜在问题和改进机会
        - 分析过程的有效性，以预测还隐藏的缺陷
        - 考察缺陷在什么阶段被引入，之前的阶段为何未能发现

项目支持活动

配置管理

- 基本概念
    - 配置项：在配置管理当中作为单独实体进行管理和控制的工作产品集合
        - 各种文档、说明书；程序代码；开发环境；产品数据文件
    - 基线：是配置项持续演进的稳定基础，发布一个基线包括该基线所有的配置项以及它们的最新变更
        - 项目开发当中比较重要的时间点，和里程碑类似
- 配置管理
    - 目的是建立与维护工作产品的完整性
    - 活动
        - 识别和记录配置项的物理特性和功能特性
            - 要有数量限制

        - 不是为了全面管控产品，而是要把那些修改不会产生较大影响的部分忽略，管控那些一定不能轻易改动的部分
    - 控制上述特性的变更
    - 记录和报告变更过程和相应的配置项状态
    - 验证配置项是否与需求一致
- 配置管理活动
    - 跟踪变更请求 是进行 配置项管理的重要手段

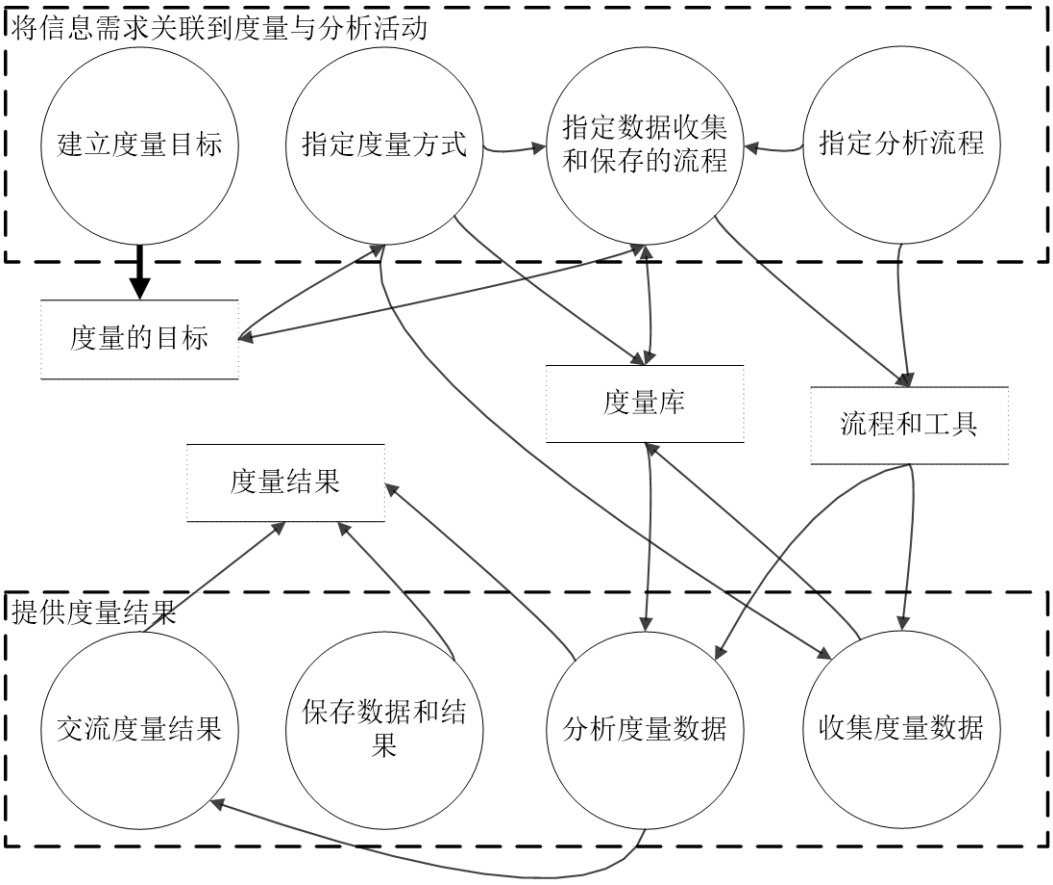


- 为什么持续集成中版本控制很重要?
    - 因为持续集成是会失败的，失败后就需要回滚版本重新来过
        - 出错后，应该把集成后发生错误的版本的代码从服务器上拉取下来
        - 然后尝试去复现错误，修改错误，编写测试证明修改成功
        - 完成之后你必须提请一个变更请求
- 三库管理★
    - 工作库
        - 所有人均可读写
        - 单元测试完成后就认为是稳定版本，移入配置库
        - 配置库和产品库的变更同步到工作库
    - 配置库
        - 由配置管理员维护，只有他能修改，其他人只能 checkout
        - 集成测试完成后移入产品库
    - 产品库
        - 版本回滚时直接回滚到产品库的上一个版本

度量和分析

- 主要联系之前的度量那一章的内容理解
- 可以支持如下的项目管理活动
    - 客观地估计与计划

    - 根据建立的计划和目标，跟踪实际进展
    - 识别与解决过程改进相关议题
    - 提供将度量结果纳入未来其它过程的基础
- 活动



- GQM
    - Goal：目标
        - 最大化所有团队贡献者的生产力
    - Question：关键问题
        - 开发人员能够完成分配给他们的任务吗，或者他们遇到障碍了吗？
    - Metric：度量
        - 由个体或者工作组产生的项目工件的数量

决策分析

- 重大决策才要走这个流程
- 活动
    - 建立评估备选方案的准则
    - 识别备选解决方案
    - 选择评估备选方案的方法
    - 使用已建立的准则与方法，评估备选解决方案
    - 依据评估准则，从备选方案中选择建议方案

根因分析

- 活动
  - 识别和选定问题
  - 根因分析（确定问题的根本原因）
  - 建立改进的行动方案
  - 实施改进，评估效果
- 鱼骨分析法
  - 鱼头是问题，鱼骨是原因
  - 每根鱼骨代表一个不同的角度，鱼骨上的小刺又是这个方面的具体原因
    - 典型角度：技术；人员；培训；过程
  - 根本原因 => 什么原因导致了这个根本原因 => （一般追问三层）

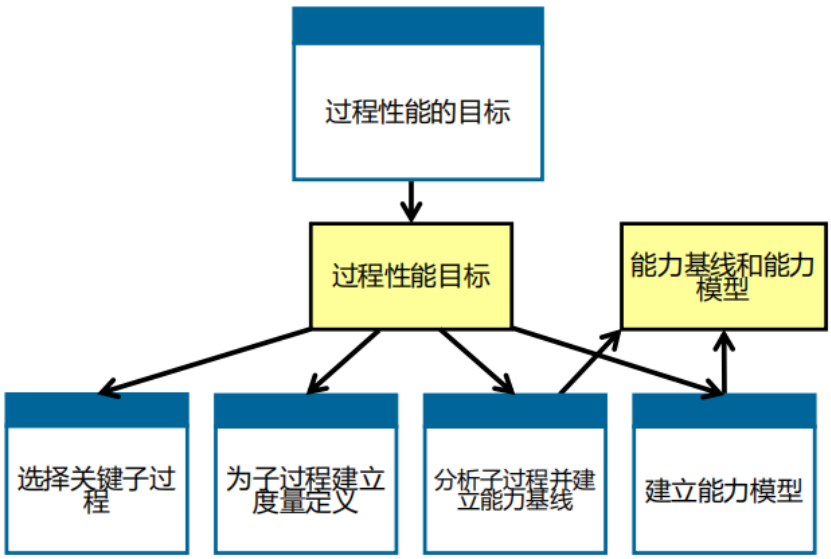
定量管理

- 回顾 CMMI
  - 4 以上都算作定量管理
  - 4 与 5 的差别
    - 4 更多关注让预测模型更加准确，让过程更加稳定（在掌控之内）
    - 5 更多关注过程改进（更上一层楼）
- 一般的项目管理
  - 关心当前状况（不等于不使用数据，当前状况也可能使用数据）
  - 但不能回答预测相关的问题
    - 维持现状，项目最后能否成功？
    - 如果在某些方面进行某些调整，会有什么不同的结果？
- 定量管理
  - 能够理解偏差
    - 为什么要理解偏差？
    - 因为有时候会面临这样的问题：客户希望 10 周交付，历史数据表明类似的任务我们能在 9~11 周完成，那么是否应该接受这个任务？
  - 定量管理必须有模型的支持

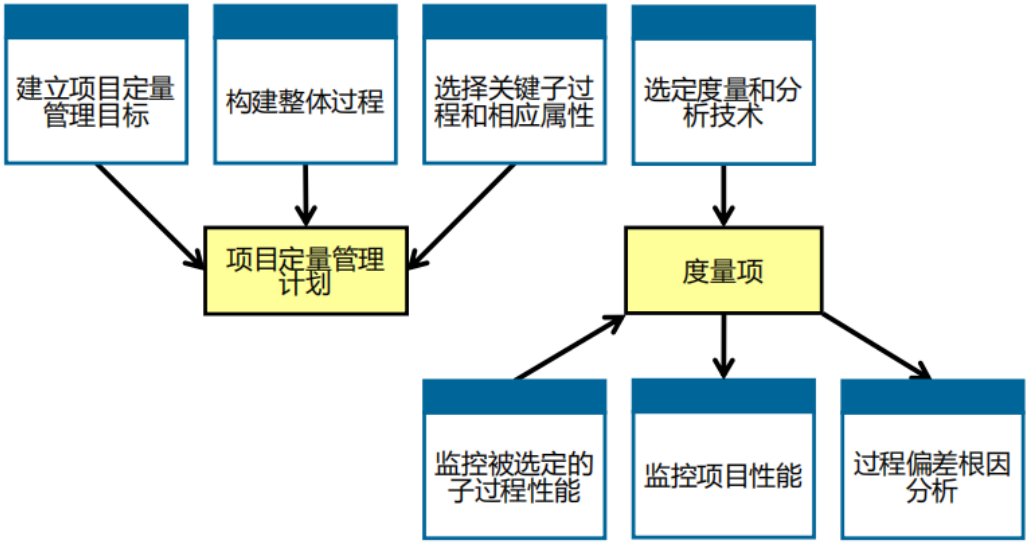
定量管理基本范式

- 过程度量
  - X_i
  - 例如：时间；缺陷消除效率
- 产物度量
  - Y_i
  - 例如：缺陷密度；相应时间
- （子）过程性能
  - 遵循某个特定过程后产生结果的量化描述
  - 既包括 X_i，也包括 Y_i
- （子）过程性能基线
  - 过程性能的定量化刻画
  - 均值 + 范围（一般使用 ±标准差）
- 过程性能模型

    - 描述子过程性能基线和整体过程有意义的输出（如质量、生产效率、成本）之间关系的数学模型
    - $Z_i = aX_1 + bY_1 + c$
    - 例如：Process Yield；Phase Yield
- 构建定量模型的关键实践



- 应用定量模型（定量管理）的关键实践



定量管理技术

非统计技术

- 描述数据集整体特征或关联关系，从而帮助选择统计技术
- 调查通过数据分析检测到异常的原因
- 直方图
    - 案例
        - 人力资源分配 / 月
        - 缺陷占比 / 阶段
- 帕累托图
    - 从高到低排列的直方图（上方有累计百分比的折线）
    - 案例
        - 缺陷占比 / 问题原因

- 因果图
  - === 鱼骨图
- 散点图
  - 表现的是一个变量相对于另一个变量的表现情况
  - 正相关 => 弱正相关 => 不相关 => 弱负相关 => 负相关

统计技术

- 量化决策的不确定条件；并指导行动，减少不确定性
- 控制图
  - 监测生产过程是否处于控制状态
  - 三条线：中心线；上控制限；下控制限
  - 中间是抽样点的折线图
  - 正态分布对其的意义
    - 如果产品质量服从正态分布，落在 $\mu \pm 3\sigma$ 内的概率为 99.37%
  - 不受控的典型情况（从上到下分为 ABCCBA 六个区）
    - 1 点落在 A 区以外
    - 连续 9 点落在中心线同侧
    - 连续 6 点递增or递减
    - 连续 14 相邻点在中心线上下交替
    - 连续 3 点中有 2 点落在中心线同侧的 B 区以外
    - 连续 5 点中有 4 点落在中心线同侧的 C 区以外
    - 连续 15 点在 C 区中心线上下
    - 连续 8 点在中心线两侧，但无一在 C 区中
  - 两类错误
    - 虚发警报错误
      - 出上下线的部分，记为 α
    - 漏发劲爆错误
      - 产品分布偏离中心线，但抽样正好抽到上下线内的部分，记为 β
- 回归分析
  - 建立自变量和因变量之间的回归方程，并将其作为预测模型
  - 找到主要因素和其数量资料，就可以采用回归分析法进行预测
  - 步骤
    - 选择变量和模型
    - 变量相关性分析
      - 相关系数 $|r|$
        - 完全 - 显著 - 高度 - 中度 - 低度 - 极弱 - 不相关.....
        - 1　　0.95　0.8　0.5　0.3　　0
    - 建立多变量回归模型
    - 检验模型（如：方差分析）
    - 模型性能分析（如：平均误差率）
- 假设检验
  - 用来判断样本与样本，样本与总体的差异是由抽样误差引起还是本质特征差别造成
  - 原理：先对总体特征做出某种假设，然后通过抽样研究的统计推理，对此假设应拒绝还是接受作出推断

    - 拒绝域；置信水平（原来就没学会，给了）
- 方差分析
    - 是采用数理统计的方法对所有结果进行分析，鉴别各种因素对研究对象的某些特征值影响大小
    - 用于两个及两个以上的样本均数差别的显著性检验
    - 目的
        - 找出对该事物有显著影响的因素
        - 研究各因素之间的相互作用是否对事物造成影响
- 仿真建模
    - 是一种计算机化的静态模型，代表某种动态系统或现象
    - 当操纵真实系统的成本、风险过高时，这是一种重要的廉价方法
    - 复杂性
        - 系统的不确定性和随机性
        - 动态行为
        - 反馈机制
    - 软件过程仿真模型
        - 关注于特定的软件开发/维护/演化过程
        - 它可以表示当前已实施（按现状）或计划在未来实施的过程
    - 好处
        - 有助于决策
        - 有助于降低风险
        - 帮助在战略、战术、运营层面进行管理
    - 目的
        - 战略管理
        - 规划
        - 控制和运营管理
        - 流程改进和技术采用
        - 理解
        - 培训和学习
    - 步骤
        - 确定建模的范围
            - 该模型的范围必须足够大，以完全解决提出的关键问题
        - 定义结果变量
            - 结果变量是回答关键问题所需的元素
            - 例如：成本；时间；人员使用率；产量
        - 过程抽象
            - 确定过程的关键元素
            - 例如：关键活动；主要对象（代码，设计等）；重要资源（人员，硬件等）；活动之间的对象流和排序
        - 选择和定义输入参数
            - 根据结果变量和过程抽象来确定
            - 例如：工作量（LOC）；发现缺陷的效率；资源限制

常用仿真技术

> 完全没听，建议放弃

- 蒙特卡罗模拟
  - 当要模拟的对象本身具有某种概率特征时，可以用计算机模拟结果
  - 步骤
    - 对每个重要变量建立一个概率分布
    - 对上述的每个变量建立累计概率分布
    - 为每个变量建立随机数区间
    - 生成随机数
    - 进行一系列模拟试验
- 离散事件仿真
  - 模拟的系统的状态变量随一个个事件的发生而在特定的时间点离散变化
  - 系统的状态变化是由（往往是随机发生的）事件驱动的
  - 应用场景
    - 适合于以过程为基础的场景
    - 有强大的模拟各种事件流、过程流的能力，并且能够定量地评价过程能力
  - 步骤
    - 系统建模
    - 确定仿真算法
    - 建立仿真模型
    - 设计仿真程序
    - 输出仿真结果
- 系统动力学仿真
  - 系统的状态随时间连续变化
  - 动态地使用一组不同的方程计算系统或过程随着时间的变化的趋势
  - 应用场景
    - 本质上不是随机的
    - 主要用于从比较高阶的层面对过程进行建模，没有关注太多的细节
  - 步骤
    - 分析阶段：运用系统动力学方法来分析研究对象，了解系统的整体结构和系统变量之间的相互作用，来解答研究的问题
    - 反馈机制分析阶段：划分系统的层次结构，了解系统整体和局部的反馈机制，确定变量的类型及变量之间的相互作用，从而明确模型的主回路
    - 建立方程式：列出各变量规范的数学方程式，并赋予常数、表函数等相关变量的初始值
    - 仿真模拟与政策分析：运行建立好的系统动力学模型，利用案例模拟仿真，并结合研究领域进行政策分析，进一步修改和完善系统
    - 模型的检验和评估：利用其他一些案例资料，对该模型进行检验模拟，然后根据结果对模型进行评价
- 混合仿真
  - 通常混合离散事件仿真和系统动力学仿真
  - 垂直分层机制
    - 位于较高层抽 象层次的仿真范式主要负责战略决策或环境动态因素的建模
    - 位于低层次的仿真范式主要用于构建操作层或具体实现活动 (编码、集成、测试等) 的建模
  - 水平联动机制

- 不同仿真范式间没有明显的界限，混合在一起
- 迭代式软件开发过程可以采用此混合机制来建立混合仿真模型